Cluster Computing (2026) 29:197
https://doi.org/10.1007/s10586-026-05944-x

Load-aware predictive auto-scaling framework for cloud environments

L. Zanussi[1] · D. Tessera[1] · L. Massari[1] · B. Bermejo[2] · C. Juiz[2] · M. Calzarossa[1]

Received: 8 May 2025 / Revised: 28 September 2025 / Accepted: 12 January 2026 / Published online: 8 March 2026

Abstract
Cloud has become an increasingly popular computing paradigm because of its benefits in terms of scalability, flexibility and cost. To make cloud solutions competitive, it is important to exploit their elasticity and dynamically adjust cloud resources in a timely manner to cope with the incoming workloads. This paper proposes a load-aware predictive auto-scaling framework that tries to anticipate workload changes and ensure at the same time the desired utilization level of the cloud resources. To this aim, we forecast the future workloads and the cloud resources necessary to cope with these workload demands. The framework has been extensively tested in a simulation environment based on the CloudSim toolkit. Synthetic and real workloads characterized by different arrival patterns have been considered. The results showcase the effectiveness of the proposed policy in scaling cloud resources according to the predicted arrival patterns.

Keywords Cloud computing · Auto-scaling policy · Workload prediction · Resource management · Scheduling · CloudSim · Virtual Machines

1 Introduction

Cloud computing has become a widespread solution for all types of businesses thanks to its flexibility, scalability and cost savings. Companies, organizations and even individuals benefit from cloud infrastructures and services for their daily activities. According to Gartner [1], by 2028 cloud computing will shift from being a technology disruptor to becoming a necessary component for maintaining business competitiveness. To cope with these issues, it is compelling to avoid resource over-provisioning or under-provisioning because of their negative effects in terms of extra monetary cost and performance degradation, respectively. Hence, to provision the "right" amount of cloud resources, an effective resource management is required.

In this context, auto-scaling mechanisms play a vital role in that they automatically exploit the elasticity typical of cloud environments by dynamically adjusting the number of allocated resources, e.g., Virtual Machines (VM), containers, as a function of the workload being processed [2, 3]. As outlined by these surveys, auto-scaling problems are often abstracted as a Monitoring, Analysis, Planning, and Execution (MAPE) control loop [4]. During the Monitoring phase, measurements of the performance metrics selected for driving the scaling actions are collected. The Analysis phase refers to the processing of these metrics to assess whether a scaling action has to be triggered. The details of the scaling, that is, the amount of resources involved, are estimated during the Planning phase. Finally, the Execution phase refers to the actual allocation or deallocation of the resources.

Notably, scaling actions are triggered by diverse criteria associated with actual or predicted performance metrics, such as usage of the resources currently allocated or desired Quality of Service (QoS) levels. For example, a scaling

✉ L. Massari
luisa.massari@unipv.it

L. Zanussi
luca.zanussi01@universitadipavia.it

D. Tessera
daniele.tessera@unipv.it

B. Bermejo
belen.bermejo@uib.es

C. Juiz
cjuiz@uib.es

M. Calzarossa
mcc@unipv.it

[1] Department of Electrical, Computer and Biomedical Engineering, University of Pavia, Pavia, Italy

[2] Department of Computer Science, University of Balearic Islands, Palma, Spain

out, i.e., an increase of the number of allocated resources, could be performed whenever the response time experienced by user requests or the resource utilization exceed some predefined thresholds. On the contrary, a scaling in, i.e., a decrease of the number of allocated resources, could be performed when resources are under-utilized, thus no longer needed. Some auto-scaling mechanisms react only after changes of the performance metrics being considered have been detected. Other mechanisms try to anticipate potential performance issues by predicting the future workload behavior and dynamically adjusting the resources in a timely manner.

This paper addresses the challenges related to scaling cloud resources under variable workload conditions. We propose a load-aware lightweight predictive auto-scaling framework to proactively allocate or deallocate resources. The core of the approach is represented by two predictive components that forecast the future workloads and the cloud resources necessary to cope with these workloads and ensure at the same time the desired utilization level. More precisely, the number of resources to be allocated or deallocated is based on an analytical formulation that takes into account the predicted workload. This represents the main innovation of our framework.

Experimental results, based on simulations, have clearly demonstrated the effectiveness of the proposed framework which quickly adjusts the resources to the predicted workload patterns.

The primary contributions of this work include:

- Load-aware auto-scaling framework that predicts when and to what extent scaling actions have to be performed;
- Extensions of the CloudSim simulation toolkit to model the proposed framework;
- Comprehensive experimental evaluation of the proposed framework under different workload patterns.

The paper is structured as follows. After a review of the state of the art, given in Section 2, we present the proposed auto-scaling framework in Section 3. The settings and the results of the experiments are analyzed and discussed in Sections 4 and 5. Finally, in Section 6 we provide some concluding remarks and outline possible future research directions.

2 Related work

Managing the dynamics of cloud environments has been (and still is) one of the main concerns of researchers and practitioners (see, e.g., [5, 6]). In this context, auto-scaling mechanisms play a key role. Several works have appeared in the literature (see, e.g., [7–10] for detailed surveys). In general, papers are classified based on the proposed approach, namely, reactive, proactive or hybrid. In what follows, we review the state of the art in these areas.

2.1 Reactive approaches

As already mentioned, reactive auto-scaling policies respond to changes of some pre-defined performance metrics after these changes have occurred. These policies are often driven by rules based on thresholds associated with consumption of cloud resources, such as CPU and memory utilization, or with applications, such as response time.

For example, the two policies for scaling out resources presented in [11] use the resource utilization level and the request response time as scaling indicators, respectively. In detail, scaling is triggered whenever the average CPU utilization of all VMs or the 95th percentile of the response time go beyond a predefined threshold. Two thresholds associated with a high order quantile of the execution time and with its mean are used in [12] for scale out and scale in decisions, respectively. Similarly, in [13] scaling actions are triggered whenever the VM utilization exceeds an upper threshold or falls below a lower threshold. The effects exercised by different incoming workload patterns on the behavior and performance of the proposed auto-scaling policies as well as the role of their configuration parameters, such as time interval for computing the individual VM utilization, are also investigated.

We remark that auto-scaling approaches based on static thresholds can suffer from the problem of oscillations in adding and removing resources, thus they are not suitable for highly unpredictable workloads.

In the literature, this problem has been addressed under different perspectives. Some studies propose solutions to adapt thresholds to workload changes. For example, in [14] a fuzzy-based approach is adopted for automatically adjusting thresholds according to the load, response time and CPU utilization.

Other studies assess to what extent the choice of the thresholds affects the performance of the auto-scaling policies. In [15] authors propose a strategy to adaptively computing the amount of resources to be allocated or deallocated according to their current utilization level and evaluate the impact of the upper and lower utilization thresholds.

An interesting approach for adding or removing servers as to maintain the right amount of spare capacity able to absorb unpredictable changes in load conditions is presented in [16]. To this aim, the policy checks the system conditions at fixed time intervals. Notably, the required capacity is determined using the total load of the system, which in turn depends on number of requests in the system when the policy is invoked, and the load a single server can handle.

2.2 Proactive approaches

Proactive auto-scaling approaches have been studied to overcome some of the limitations of the reactive approaches (see, e.g., [17–24]). In fact, by predicting the resources required by the workload, it is possible to anticipate scale out and scale in actions, thus reducing the leasing cost of the VMs as well as improving the workload performance.

A model-driven auto-scaling approach for multi-tier cloud systems is presented in [17]. More specifically, to infer the amount of resources required to ensure response time guarantees, the proposed modeling engine leverages a product form queueing network model that relies on the Kalman filter to estimate model parameters that cannot be monitored. A scaling action is triggered after two successive directives from the modeling engine.

In [18], scaling actions are based on predictions of resource usage. These predictions are obtained from the periodic patterns identified in the time series describing resource usage or, in case no patterns exist, from a Markov chain whose states represent groups of resource usage samples. More specifically, the time series is divided into fixed duration segments whose size is determined through spectral analysis. A pattern is then inferred from the presence of highly correlated segments. As an alternative, predictions of resource usage are drawn from the state transition probabilities of the Markov chain.

A policy that employs a receding horizon control technique is proposed in [19]. For each step ahead in the time horizon, the proposed policy predicts incoming workload using a second-order Auto-regressive Moving Average model and iteratively computes the cost of all possible resource allocations using the results provided by the Mean Value Analysis. The optimal resource provisioning is then obtained by solving a single-objective optimization problem that minimizes the overall allocation cost.

It is important to outline that accurate predictions of resource requirements rely on workload models able to characterize and forecast future workload demands (see, e.g., [25–29]). As an example, the predictive model presented in [27] is composed of three layered neural networks. This model learns and extracts patterns from the workload to be used for future predictions.

2.3 Hybrid approaches

Many papers propose hybrid auto-scaling approaches that combine the benefits of proactive and reactive approaches (see, e.g., [30–40]).

The heterogeneity of cloud resources is addressed by the auto-scaling system presented in [30]. The system includes a predictor of the future incoming workload demands, a profiler of the throughput of each VM type while running an application and a scaler. The scaler implements a reactive policy based on an upper and a lower threshold associated with some performance metrics. Scaling actions are triggered whenever the workload predictor forecasts persistent over-load or under-load conditions over a given time interval. In contrast, the estimated throughput is used by the scaler to identify the most appropriate resource combination that satisfies both the workload and customer SLA requirements.

In some papers the hybrid approaches leverage reactive policies to scale out resources and proactive ones to scale in. In other papers proactive policies are used to predict future workloads, while reactive policies manage unpredicted workload changes.

For example, in [31] the reactive approach proposed for triggering scaling out actions considers a threshold associated with the 95th percentile of the average request response time, computed using measurements periodically collected. In contrast, whenever the response time requirements are satisfied for a given number of consecutive time intervals, a proactive approach checks whether a scaling in action is necessary. To this aim, the number of required resources are predicted using a polynomial regression model.

Two adaptive controllers incorporating both reactive and proactive components are presented in [32]. To manage scaling decisions and prevent oscillations, these controllers scale the number of VMs allocated to a service running in the cloud based on the current and predicted future load on the service. In detail, the estimation of the rate of adding or removing VMs can be based on the periodical rate of change of the system load or on the load change with respect to the average capacity.

A framework for distributed system auto-scaling through the combination of proactive and reactive techniques is proposed in [33]. The framework predicts the trends of the time series describing some high-level performance metrics and uses these predictions to take proactive scaling decisions. As a contingency to possible prediction failures, scaling actions are triggered in a reactive manner using different threshold rules.

Similarly, the framework presented in [36] combines predictive and reactive methods for scaling cloud resources. A time series method, namely, an auto-regression of order one, is applied to forecast the future behavior of the system, while a queueing network model is used to compute the required number of VMs and their capacity. A reactive approach is employed to take corrective actions when the provisioned resources are insufficient to cope with an anomalous workload increase.

We remark that auto-scaling mechanisms are generally offered by cloud providers [41–43] to allow their users to

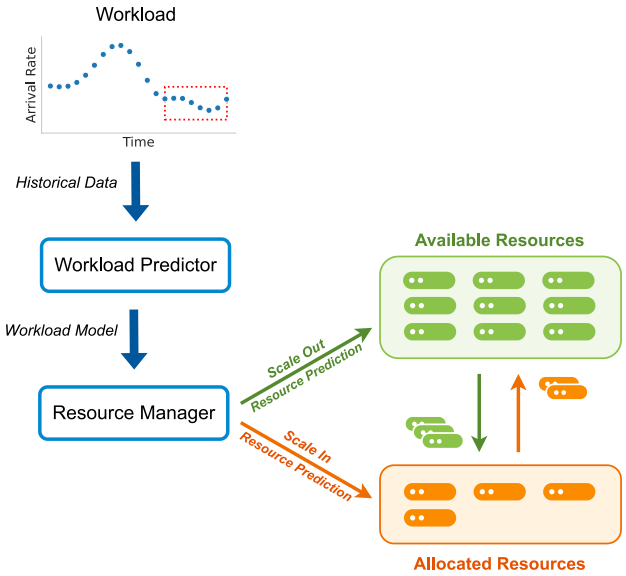

Fig. 1 Overview of the proposed auto-scaling framework

exploit cloud elasticity. These mechanisms are often reactive or proactive approaches based on rules specified by the users. For example, scaling actions can be triggered by the violation of user-defined constraints on some metrics. As an alternative, users can define a target value for a given metric and to keep this metric as close as possible to the target the auto-scaling system adds or removes VMs.

In summary, the literature offers many diverse solutions to the auto-scaling problem. To the best of our knowledge, no papers have proposed a lightweight solution that scales resources by making predictions of the future VM utilization using the system throughput, which in turn is obtained from the predictions of the incoming workloads and the current status of the system.

3 Proposed auto-scaling framework

As already mentioned, we propose a load-aware predictive auto-scaling framework whose objective is to scale out and in cloud resources as to keep their overall utilization at a desired level. To this aim, we predict the future workloads and the resources necessary to cope with these workload demands. Figure 1 provides an overview of the proposed framework.

We can identify two main components, namely:

- A workload predictor responsible of forecasting future user requests based on continuously updated historical workload data;
- A resource manager responsible of triggering scale out and scale in actions and predicting the amount of cloud resources able to cope with the predicted workload and ensure at the same time the desired utilization level.

Table 1 Summary of the main notations

δ	Time slot duration
M	Number of time steps
h	Time horizon
S	Request service demand
λ	Arrival rate
$\hat{\lambda}$	Predicted arrival rate
$\hat{N}$	Number of requests to be processed
$\hat{Q}$	Predicted request backlog
$\hat{X}$	Predicted throughput
$X^{\max}$	Maximum throughput
$\hat{C}$	Number of requests to be completed
$\hat{U}$	Predicted resource utilization
U^{ref}	Desired resource utilization
R	Number of allocated resources
$\hat{R}^*$	Predicted number of resources

It is important to remark that the framework includes another component, namely, a scheduler, that maps user requests to cloud resources according to different policies. This component has not been depicted in the figure because it does not directly affect the scaling actions.

In what follows, we describe in detail the proposed framework. Table 1 summarizes the key notations used in this section.

3.1 Workload predictor

The workload predictor aims to forecast future workloads using historical data. More precisely, starting from the arrival rates λ_i measured at M equally spaced time steps t_i, $(i = 1, 2, \ldots, M)$, we predict the arrival rate $\hat{\lambda}_j$ at time t_j, $(j = M+1, M+2, \ldots, M+h)$, h being the time horizon. These predictions are based on the model obtained by fitting the measured values.

In this context, the choice of the regression model that best fits the workload data is of paramount importance. Various regression techniques, such as linear regression, polynomial regression, Auto-Regressive Integrated Moving Average (ARIMA) and Recurrent Neural Network (RNN), can be applied to achieve this goal. Some techniques, such as polynomial regression, need only the last M historical data and are characterized by lightweight computations. In contrast, other techniques, such as ARIMA and RNN, require much larger amount of historical data to derive the model structure and coefficients. Hence, these characteristics make the polynomial regression well suited for runtime workload predictions.

Figure 2 shows an example of the model that fits the arrival rates measured every 15 seconds over an interval of five minutes. This model is a polynomial of degree two, namely, $\lambda(t) = a_0 + a_1\, t + a_2\, t^2$.

Since workload arrivals typically vary over time, predictions need to be adjusted accordingly. Hence, as soon as new workload measurements are available, the historical workload data is updated by using a rolling window approach as shown in Fig. 3. In detail, the oldest measurements are removed and new measurements are considered, thus the coefficients of the model are computed at each time step without re-estimating the model. This process is repeated continuously to ensure predictions follow as close as possible the actual workload patterns.

3.2 Resource manager

The resource manager is responsible of allocating and deallocating cloud resources. To achieve this, it predicts to what extent resources need to be adjusted to cope with the future workload and to ensure the desired utilization level.

We remark that the choice of the desired utilization level plays a key role in our framework to avoid resource overprovisioning and maintain sufficient processing capacity. This will also allow the mitigation of potential prediction inaccuracies due to fluctuating workloads.

In what follows we describe the policies proposed to trigger scaling actions and predict the corresponding amount of resources.

3.2.1 Scale out policy

As already mentioned, scale out actions refer to the allocation of additional resources to cope with an increase in the incoming user requests. We remark that in general the resources being allocated are not immediately available because of delays in their start up (e.g., boot time), thus at time t_i our policy predicts the amount of resources necessary after the start up, that is, at time t_{i+h}.

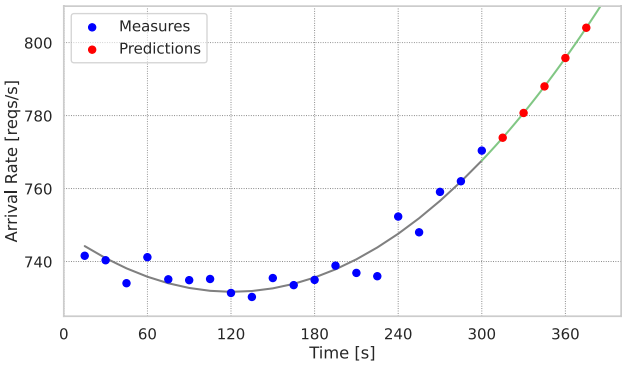

Fig. 2 Polynomial of degree two fitting the measured workload data represented by blue dots. The red dots refer to the arrival rates predicted by the model

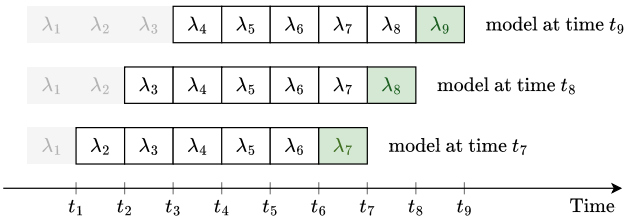

Fig. 3 Rolling window approach used to update the models of the arrival rates at three consecutive time steps. Each model is obtained by fitting six measurements

As specified by the pseudo code of Algorithm 1, the predicted utilization of cloud resources represents the core of the proposed scale out policy. We recall that, according to the utilization law, the resource utilization is given by the product of the throughput and the average service demand at the resources [44]. The service demand S is typically known as it is a characteristic of the requests that depends on the resource processing speed. On the contrary, the throughput, that is, the number of requests completed by the resources per time unit, is unknown. Moreover, we cannot assume

Require: i current step associated with time t_i
W_i rolling window of the last M measured arrival rates
Q_i number of unprocessed requests measured at time t_i
R number of cloud resources allocated at time $t_j, (j = i, \ldots, i+h)$
h number of time steps ahead for the prediction
δ time step duration
S request service demand
U^{ref} desired resource utilization
Ensure: predict the number of required cloud resources at time t_{i+h}

▷ Workload and backlog predictions

1: arrivalModel.fit(W_i)
2: $\hat{Q} \leftarrow$ backlogPrediction$(i, h, Q_i,$ arrivalModel$, R)$

▷ Resource prediction

3: $\hat{\lambda} \leftarrow$ arrivalModel.predict$(i+h)$
4: reqsToBeProcesssed $\leftarrow \hat{Q} + \hat{\lambda} * \delta$
5: predictedThroughput $\leftarrow$ reqsToBeProcesssed$/\delta$
6: predictedTotalUtilization $\leftarrow$ predictedThroughput $* S$
7: predictedResources $\leftarrow$ predictedTotalUtilization$/U^{\text{ref}}$
8: predictedNumberOfResources $\leftarrow$ Ceiling(predictedResources)

Algorithm 1 Load-aware scale out policy

steady-state conditions, that is, the throughput equals the arrival rate, because the allocated resources might be unable to process all incoming requests in a given time slot.

Hence, to estimate the throughput we have to take into account the predicted arrival rate (provided by the Workload Predictor) as well as the predicted backlog of user requests that could not be completed in the given time slot by the cloud resources currently allocated.

Algorithm 2 presents the pseudo code iterated for predicting the backlog $\hat{Q}_j$ $(j = i+1, i+2, \ldots, i+h)$. These values are obtained by combining the workload predictions with the maximum number of requests that can be processed by the allocated VMs.

More precisely, the total number of requests $\hat{N}_j$ to be processed in the time interval $[t_{j-1}, t_j]$ of duration δ is given by:

$$\hat{N}_j = \hat{Q}_{j-1} + \hat{\lambda}_j \times \delta$$

where $\hat{Q}_{j-1}$ denotes the backlog of requests at time t_{j-1}. We remark that the backlog Q_i at time t_i is not predicted as it can be easily measured on the system.

The number of requests $\hat{C}_j$ that can be completed during the given time slot of duration δ by the allocated cloud resources, is then obtained as follows:

$$\hat{C}_j = X_j^{\max} \times \delta$$

where $X_j^{\max}$ denotes the maximum throughput of the resources, i.e., the maximum number of requests that can be completed per time unit under the assumption that the utilization of each resource is equal to one. The difference between $\hat{N}_j$ and $\hat{C}_j$ gives the predicted backlog $\hat{Q}_j$ in the considered time slot. Note that $\hat{Q}_j$ is zero when the cloud resources are able to complete all the requests they have to process.

Require: Q_s initial number of unprocessed requests
$\hat{\lambda}$ predicted arrival rate
r number of cloud resources currently allocated
δ time step duration
S request service demand

Ensure: Q_e final number of unprocessed requests

```
function NEXTBACKLOG(Q_s, λ̂, r, δ, S)
    reqsToBeProcesssed ← Q_s + λ̂ * δ
    resourceMaxThroughput ← 1/S
    maxThroughput ← r * resourceMaxThroughput
    maxReqsThatCanBeCompleted ← maxThroughput * δ
    if reqsToBeProcesssed > maxReqsThatCanBeCompleted then
        Q_e ← reqsToBeProcesssed − maxReqsThatCanBeCompleted
    else
        Q_e ← 0
    end if
    return Q_e
end function
```

Algorithm 2 Prediction of the number of unprocessed requests

Since we predict at time t_i the amount of cloud resources necessary at time step $j = i+h$, such that $\hat{Q}_{j+1} = 0$, by applying the utilization law, we obtain the predicted overall utilization of cloud resources $\hat{U}_j$, that is:

$$\hat{U}_j = \hat{X}_j \times S$$

where $\hat{X}_j$ is derived as the ratio between the total number of requests $\hat{N}_j$ to be processed in a time interval of duration δ and δ. The predicted utilization is used to decide whether and to what extent a scaling action has to be performed.

Notably, the ratio $\hat{U}_j / U^{\text{ref}}$ provides a prediction of the number of resources $\hat{R}_j$ necessary at time t_j to ensure the desired utilization level U^{ref}. We remark that the result of this ratio is typically a real number, thus the ceiling function maps $\hat{R}_j$ to the least integer greater than or equal to $\hat{R}_j$, namely, $\hat{R}_j^* = \lceil \hat{R}_j \rceil$.

Depending on the relationship between the predicted number of resources $\hat{R}_{i+h}^*$ at time t_{i+h} and R_{i+h-1}, that is, the number of resources available at time t_{i+h-1}, the resource manager might trigger a scaling out. We outline that R_{i+h-1} takes into account the effects of scale out actions scheduled earlier than t_i. For example, the effects of a scale out triggered at time t_{i-2} will appear at time t_{i+h-2}.

In case $\hat{R}_{i+h}^*$ and R_{i+h-1} are equal, no scaling actions are necessary. On the contrary, a scale out is triggered when $\hat{R}_{i+h}^*$ is greater than R_{i+h-1}. The number of resources diff_{i+h} to be allocated at time t_i is given by:

$$\text{diff}_{i+h} = \hat{R}_{i+h}^* - R_{i+h-1}$$

We emphasize once more that these resources will only be available for processing incoming user requests at time t_{i+h}.

3.2.2 Scale in policy

In the context of scale in actions, the behavior of the resource manager is rather conservative as it avoids "early" resource deallocation. In fact, service providers are often interested in ensuring good quality of service levels even though this might require some resource over-provisioning. To this aim, before deallocating any resource, the resource manager checks three conditions with the main objective of identifying "persistent" over-provisioning conditions in a given time interval.

Notably, to avoid oscillations, no scale in actions are allowed whenever scale out actions are in progress in the considered time interval. In addition, no scale in actions can be triggered whenever the utilization of cloud resources exceeds the desired utilization level in any of the *h* slots of the interval.

$$\hat{R}_j^* = \lceil \hat{U}_j / U^{\text{ref}} \rceil$$

In case the condition $R_j > \hat{R}_j^*$ holds for *h* consecutive time slots, the resource manager performs a scale in action. The predicted number of over-provisioned resources $\hat{R}_j^{\text{over}}$ is given by the difference between R_j and $\hat{R}_j^*$.

We remark that to avoid aggressive actions that might affect performance, the minimum number of predicted over-provisioned resources over the *h* time slots is selected for the deallocation. Moreover, scale in actions have to ensure the threshold set for the minimum number of allocated resources to be guaranteed. We also outline that the actual deallocation of a VM occurs as soon as it has completed all the requests still in progress.

In summary, our proposal offers a flexible framework for scaling cloud resources in a timely manner according to the predictions of the future user requests and of the resources necessary to cope with these requests. This framework could be very beneficial for practitioners in that it will allow them to allocate the "right" amount of resources and ensure at the same time the desired utilization level.

Require: i current step associated with time t_i
W_i rolling window of the last M measured arrival rates
Q_i number of unprocessed requests measured at time t_i
R number of cloud resources allocated at time $t_j, (j = i, \ldots, i+h)$
h number of time steps to be checked
δ time step duration
S request service demand

Ensure: over-provisioning over all h time steps

```
arrivalModel.fit(W_i)
overResources ← +∞
for (j = i + 1, ..., i + h) do
    λ̂ ← arrivalModel.predict(j)
    reqsToBeProcesssed ← Q_{j−1} + λ̂ ∗ δ
    predictedThroughput ← reqsToBeProcesssed/δ
    predictedTotalUtilization ← predictedThroughput ∗ S
    predictedResources ← predictedTotalUtilization/U^ref
    predictedNumOfResources ← Ceiling(predictedResources)
    rDiff ← R_j − predictedNumOfResources
    overResources ← min(overResources, rDiff)
    Q_j ← 0                    ▷ Under the assumption that no resources are saturated
end for
```

Algorithm 3 Number of over-provisioned resources

The third condition specifically refers to the prediction of resource over-provisioning over *h* consecutive time slots. In detail, as specified by the pseudo code of Algorithm 3, at each time step j $(j = i+1, \ldots, i+h)$ the resource manager predicts the overall utilization $\hat{U}_j$ as the product of the predicted throughput $\hat{X}_j$ and the request service demand *S*. This computation is performed under the assumption that all requests $\hat{N}_j$ to be processed in the given time slot are actually completed.

Similarly to the approach used for scale out actions, the predicted amount of resources $\hat{R}_j^*$ that will ensure the desired utilization level U^{ref} at time t_j is given by:

4 Experimental setup

This section focuses on the setup of the experiments carried out to assess the performance and the effectiveness of the proposed auto-scaling policy. More precisely, we describe the main characteristics of the simulation environment developed for this aim as well as the characteristics of the workloads and of the cloud infrastructure considered in the

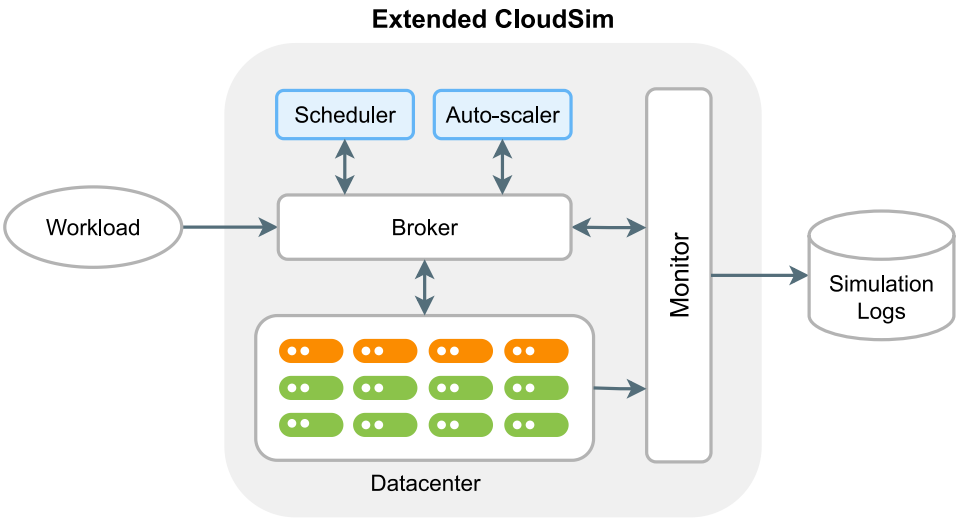

Fig. 4 Architecture of the proposed simulation environment

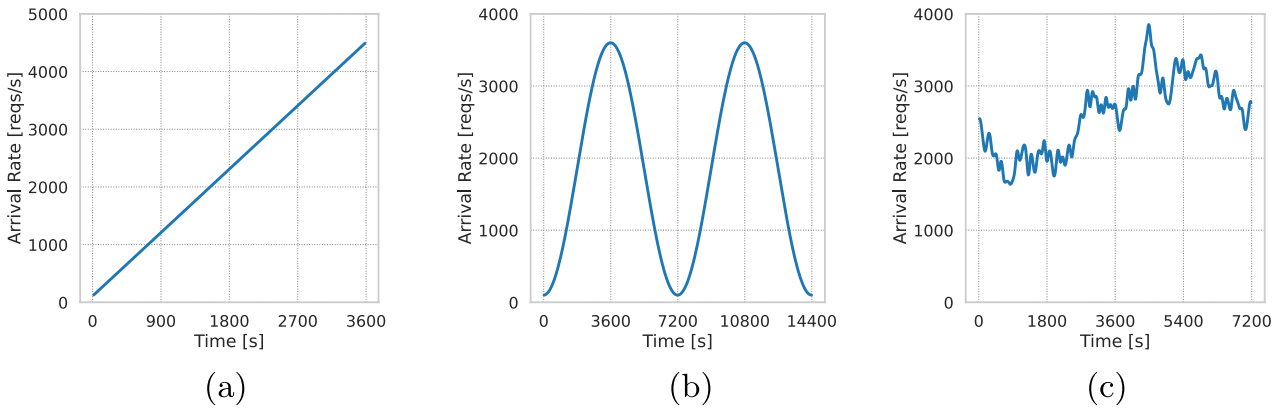

Fig. 5 Increasing (a), periodic (b) and unpredictable (c) synthetic workload arrival patterns considered in the simulation experiments

experiments. The specific settings of the simulations are also reported.

4.1 Simulation environment

To evaluate the proposed auto-scaling framework, we developed a simulation environment based on the CloudSim simulation toolkit [45]. Figure 4 shows the architecture of the proposed environment. The core of the architecture is represented by a broker whose role is the orchestration of all the other simulation components.

Notably, the broker receives the workload, i.e., user requests, and interacts with the cloud resources located in the datacenter. These interactions are driven by two components, namely, a scheduler and an auto-scaler, implemented as CloudSim extensions.

In detail, the scheduler provides the mappings between the user requests and the cloud resources allocated in the datacenter. To this aim, different policies, such as round robin, weighted round robin, can be used. The auto-scaler implements the auto-scaling policy and provides the broker with the amount of resources to be allocated or deallocated as to satisfy the desired utilization level.

Table 2 Main characteristics of the synthetic workloads considered in the simulation experiments

	Arrival rate [reqs/s]		Number
	Minimum	Maximum	of requests
Increasing	100	4,500	8×10^6
Periodic	100	3,600	26×10^6
Unpredictable	1,600	3,850	19×10^6

To keep track of the status of the system, the broker relies on the monitoring component implemented as another CloudSim extension. This component continuously collects from the datacenter measurements related to the cloud resources and the workload requests being processed. For example, measurements refer to the types of events (e.g., request arrivals and completions, resource allocations and deallocations) and their timestamps.

The measurements being collected are also stored into a log file for offline processing, that is for a detailed analysis

of the behavior of the auto-scaling policy and for computing metrics summarizing its performance.

4.2 Workload and infrastructure characteristics

The workloads considered in the simulation experiments are described in terms of time-varying arrival patterns. In detail, we generate three synthetic workloads characterized by different arrival patterns, namely, increasing, periodic and unpredictable. The selection of these patterns is motivated by their ability to stress our auto-scaling policy under a continuous increase of the arrival rate as well as under regular or sudden changes between increasing and decreasing rates. Figure 5 depicts the patterns of the three synthetic workloads, while Table 2 summarizes their main characteristics.

As can be seen, the arrival rate of the increasing workload linearly grows from 100 up to 4,500 reqs/s over one-hour interval, while the periodic workload consists of two periods of a sinusoidal pattern, each ranging from 100 to 3,600 reqs/s over a four-hours interval. The arrival rate of the unpredictable workload changes over a two-hours interval, from 1,600 up to 3,850 reqs/s without following any predefined pattern.

We remark that these workloads are composed by requests with identical characteristics, that is, processing demand equal to 10 Million instructions and service demand equal to 10 ms on the simulated infrastructure.

In addition to the three synthetic workloads, we test our auto-scaling policy using a real workload based on a flight seats availability service [46]. The workload data has been obtained by monitoring a set of consolidated VMs running on a single physical machine and sharing all its hardware and software resources. This system can receive several million transactions per hour.

As shown in Figure 6, this workload follows a diurnal pattern with arrival rates ranging from 600 up to 15,800 reqs/s over 24 hours. Requests are characterized by processing demands between 6.3 and 15.7 Million instructions and service demands between 6.3 ms and 15.7 ms on the simulated infrastructure. Figure 7 plots the empirical distribution of the processing demands of the 693 million requests of this workload. We remark that the processing demand of about 99.5% of requests is between 8 and 14 Million instructions.

The simulated cloud environment consists of an unlimited pool of Virtual Machines (VMs) whose processing speed is set to 1,000 MIPS. Each VM is also characterized by a boot time, set to 60 seconds in our experiments, that accounts for the VM startup. In detail, boot time represents the delay between the time a VM is provisioned and the time it becomes available to process incoming requests. In contrast, as already mentioned, VM deallocation is affected by delays related to the completion of previously assigned requests.

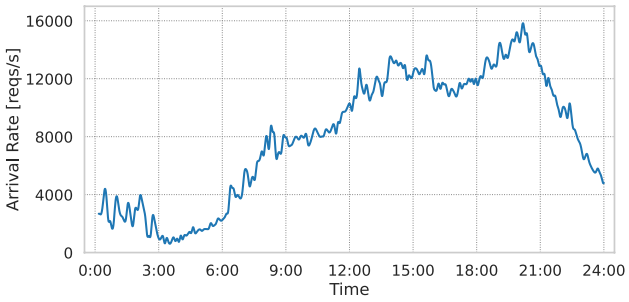

Fig. 6 Workload arrival pattern of a flight seats availability service measured over 24 hours

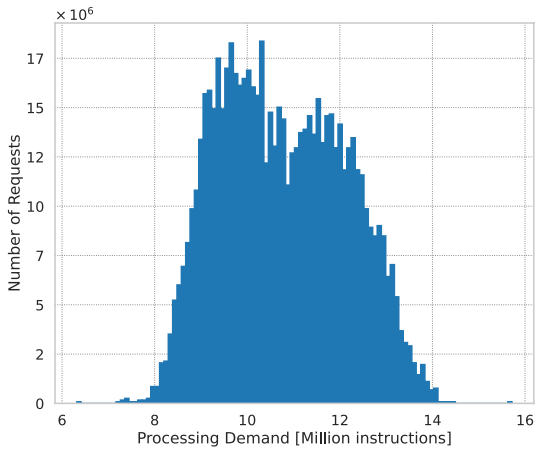

Fig. 7 Empirical distribution of the processing demand of the requests of the real workload

Table 3 Simulation parameters

Time slot interval [s]	15
Prediction history [s]	300
Prediction horizon [s]	60
Utilization level	0.8
Minimum number of VMs	2
Simulation warm-up [s]	600

4.3 Simulation settings

Table 3 summarizes the main settings used for the simulation experiments. More specifically, we set the time slot, i.e., the time between two consecutive invocations of the auto-scaling policy, to 15 seconds, whereas for the workload predictions we set the history to 300 seconds, corresponding to workload data collected over 20 consecutive time slots, and a time horizon of 60 seconds, i.e., four time slots, corresponding to the VM boot time. We recall that historical workload data are updated according to a rolling window approach.

We remark that the workload predictions are based on a polynomial regression. To identify the degree of the polynomial that best fits the workloads considered in our experiments, we apply the forward nested model selection technique. In addition, to avoid oscillations due to overfitting, we limit the degree of the polynomial to five. As a result we obtain a polynomial of degree two.

Moreover, the desired utilization level of the VMs is set to 0.8. During the simulation, two is the minimum number of VMs that must be always available for processing the incoming requests, regardless of any scaling decision. Finally, to eliminate transient effects from the collected measurements, a 600 seconds warm-up time is considered.

5 Experimental results

This section showcases the proposed auto-scaling framework by assessing its performance and effectiveness through extensive simulation experiments. We first introduce the metrics used for these assessments and the policies considered as baselines. We then analyze and discuss the simulation results.

5.1 Metrics and baseline policies

The effectiveness of the proposed auto-scaling policy is evaluated by considering metrics related to the VMs provisioned to process the incoming user requests and to the QoS being offered. In detail, an important metric considered for our assessments is the cost of the VMs being provisioned, measured as their total leasing time. This metric is of particular interest for service providers who are willing to minimize the cost for their services to stay competitive.

We also consider the response time of the requests, that is, how long each request takes to be completed. In fact, providers have to ensure good quality levels and avoid violations to the levels agreed with end users.

In addition, to estimate the overhead introduced by our auto-scaling policy, we measure the time spent by all invocations of the policy and the fraction of invocations leading to a scaling action.

As baselines for assessing the benefits of our Load-Aware policy, we choose a reactive auto-scaling policy and two static resource provisioning approaches. We remark that we did not consider any proactive/hybrid approach because their peculiarities make their implementation difficult and not always feasible.

In detail, the reactive policy, namely, the Average Load policy proposed in [13], adjusts the number of allocated VMs based on the average utilization levels of each VM, computed over a 75 second time window. This policy performs scale out or scale in actions whenever the utilization of at least one VM is above or below the corresponding thresholds, set to 0.8 and 0.4, respectively.

In contrast, static provisioning allocates a fixed number of VMs which remain always on independently of the workload arrivals. More precisely, we consider two static provisioning approaches, where the number of allocated VMs is derived according to the middle load (MID) and to the maximum load (MAX).

5.2 Simulation results

This section summarizes the results of the simulation experiments performed to assess the effectiveness of our auto-scaling policy under different workloads. We first focus on the comparisons with the selected baselines under the

Table 4 Comparison of the performance of our auto-scaling policy with the Average Load policy and the two static provisioning approaches chosen as baselines for the three synthetic workloads

	VMs		Scaling Actions	Leasing Time [min]	Response Time [ms]		
	Min	Max			Mean	Std dev	Max
Increasing							
Load-Aware	11	57	45	1,743	10.2	0.5	26.5
Average Load	11	50	39	1,545	10.7	1.2	36.3
MID	29	29	–	1,450	85,162.2	111,814.7	372,721.3
MAX	57	57	–	2,850	10.1	0.2	17.3
Periodic							
Load-Aware	2	45	169	5,751	10.3	1.0	97.3
Average Load	2	46	173	6,302	135.4	415.9	2,554.5
MID	24	24	–	5,520	459,297.3	363,901.2	974,606.3
MAX	45	45	–	10,350	10.1	0.3	61.6
Unpredictable							
Load-Aware	21	50	213	3,825	10.3	0.7	33.7
Average Load	31	44	13	4,179	10.1	0.4	22.7
MID	33	33	–	3,960	2,681.5	7,139.9	32,020.4
MAX	49	49	–	5,880	10.1	0.2	14.3

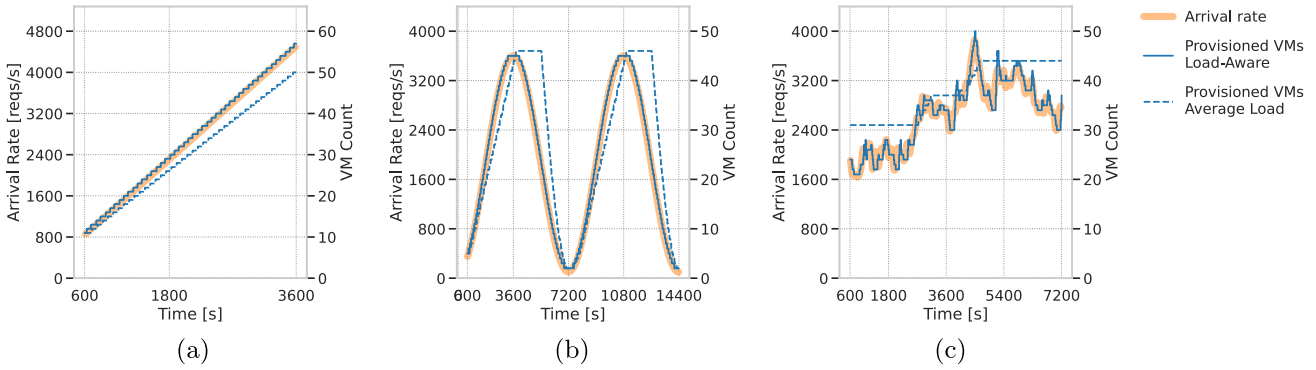

Fig. 8 Workload arrival rate (orange line) and number of VMs allocated by the Load-Aware (blue solid line) and Average Load (blue dashed line) policies for the increasing (a) periodic (b) and unpredictable (c) workload patterns

synthetic and real workloads. We then consider the real workload to investigate the performance under different utilization levels of the allocated VMs.

5.2.1 Synthetic workloads

To gain some preliminary insights into the benefits of our auto-scaling policy, we simulate the synthetic workloads described in Sect. 4.2 with our policy as well as with the three policies selected as baselines.

Table 4 compares the performance of these policies.

As can be seen, our policy generally outperforms the three baselines. For example, as expected, the comparison of our policy with the MID provisioning highlights significant differences in the response times of the requests, even though for the increasing and periodic patterns the leasing times of the VMs provisioned by MID are slightly lower, i.e., 16.8% and 4% lower, respectively. In contrast, compared to the MAX provisioning, our policy always performs much better in terms of leasing time with similar values of the request response times.

In addition, the comparisons with the Average Load reactive auto-scaling policy clearly demonstrate the benefits of our policy even in the case of the unpredictable workload pattern where predictive policies often fail. Figure 8 shows the arrival rates of the requests together with the number of allocated VMs as a function of the simulated time for the three synthetic workloads. As can be seen, unlike the Average Load policy, the number of VMs allocated by our policy closely follows the three different workload patterns. In fact, thanks to the workload predictor, our policy forecasts future changes in the arrival rates and adjusts cloud resources to ensure the desired utilization level. On the contrary, the Average Load policy only reacts to previous changes, thus delaying the scaling actions with a potential increase of the response time and of the VM leasing time.

This phenomenon is particularly evident for the periodic workload. In fact, the timely scaling actions of our policy yields a leasing time about 10% shorter than that experienced by the Average Load policy. In addition, the delayed scaling out actions of this policy lead to resource saturation, thus making the mean response time more than one order of magnitude larger.

Table 5 Performance of our policy and of the Average Load policy under the real workload

	Load-aware	Average load
Leasing Time [h]	2,627	2,994
Scaling Actions	1,567	362
Effective Invocations	27.4%	6.3%
Response Time [ms]		
Mean	10.8	210.3
Std dev	1.4	2,813.4
Min	6.3	6.3
Max	52.3	51,657.0

In terms of scaling actions, our policy and the Load-Aware policy perform a similar number of actions for all but the unpredictable pattern. In fact, to adapt to this challenging pattern, our policy keeps adjusting its predictions and the number of allocated VMs, thus it performs more than 200 actions that lead to a shorter leasing time and a comparable response time with respect to the Average Load policy.

We can then conclude that our policy is more robust to changes in arrival patterns than the considered baselines.

We remark that to run the simulation experiments, we use a workstation equipped with an Intel Core i7-9700K processor at 3.6 GHz, 64 GB of RAM and 1 TB of solid state drive and running Fedora Linux 41 as operating system. Our simulations are quite fast. For example, an experiment of our policy under the periodic workload takes about six minutes. We also measured the time spent at each time slot to predict the arrival rate and estimate the number of VMs to be allocated or deallocated. This time was of the order of few milliseconds (i.e., on average 3 ms). Even though these operations are executed repeatedly, that is, every 15 seconds, the overhead introduced by our policy is negligible.

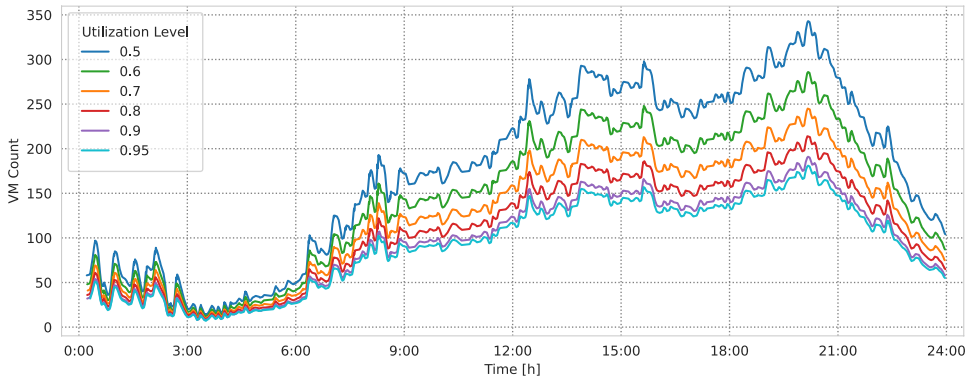

Fig. 9 Number of allocated VMs over 24 hours for different values of the desired utilization level

Table 6 Response time distribution, expressed in milliseconds, as a function of the utilization level

Response time	Utilization level					
	0.50	0.60	0.70	0.80	0.90	0.95
Min	6.3	6.3	6.3	6.3	6.3	6.3
1 st quartile	9.7	9.7	9.7	9.7	9.9	10.4
Median	10.7	10.7	10.7	10.8	11.2	11.8
3rd quartile	11.9	11.9	11.9	11.9	12.4	13.3
90th percentile	12.7	12.7	12.7	12.8	13.4	15.3
95th percentile	13.1	13.1	13.1	13.2	14.1	16.9
99th percentile	13.7	13.7	13.7	13.9	16.3	20.8
99.9th percentile	14.1	14.1	14.3	15.8	20.7	29.1
Max	23.2	30.3	31.8	52.3	101.5	3,127.9

5.2.2 Real workload

To further investigate the effectiveness of our policy in a realistic scenario, we consider the real workload whose characteristics are described in Sect. 4.2.

Table 5 compares the performance of the Load-Aware and the Average Load policies under this workload.

Similarly to what observed for the synthetic workloads, we notice that our policy significantly outperforms the Average Load policy. Notably, the total VM leasing time of our policy, equal to 2,627 hours, is 12% shorter than the Average Load counterpart. This is because our policy performs more than 1,500 scaling actions to timely adapt the number of allocated VMs to the workload patterns. This behavior is also beneficial for the request response times whose mean value is one order of magnitude shorter than the Average Load counterpart. In fact, the slow reactions to workload changes of the Average Load policy lead to VM saturation conditions, thus resulting in very large response times.

5.2.3 Impact of utilization level

Another set of experiments refers to the evaluation of the impact of the VM utilization level set in our policy to trigger scaling actions. To this aim, we simulate the real workload varying the utilization levels between 0.5 and 0.95. We remark that low VM utilization levels prevent resource saturation due to workload prediction errors or sudden unpredictable workload raises even though these advantages come at the expense of an increased number of VMs and leasing times.

Figure 9 offers an overview of the number of allocated VMs as a function of the time of the day.

As can be seen, all patterns follow the diurnal behavior of request rate, even though there are significant differences across the various utilization levels. Notably, when the desired utilization level is low, the policy tends to allocate a larger number of resources (e.g., as many as 343 for utilization equal to 0.5), thus leading to over-provisioning conditions. On the contrary, for utilization levels equal to 0.9 and 0.95, the policy allocates much fewer resources (e.g., as few as 181 for utilization 0.95), thus leading to under-provisioning conditions. For example, on average 175 VMs are allocated over the 24 hours when the utilization is set to 0.5, whereas only 97 and 92 for utilization set to 0.9 and 0.95.

As Table 6 suggests, the request response time is clearly affected by these conditions.

As expected, under-provisioning conditions generally lead to an increase of the response time. These effects are particularly evident for the values of the utilization equal to 0.9 and 0.95. In fact, the median response times, obtained for a utilization up to 0.8, only slightly differ, whereas the corresponding values for the utilization equal to 0.9 and 0.95 are about 4.7% and 10.3% greater, respectively. We can notice the same behavior up to the 99th percentile with

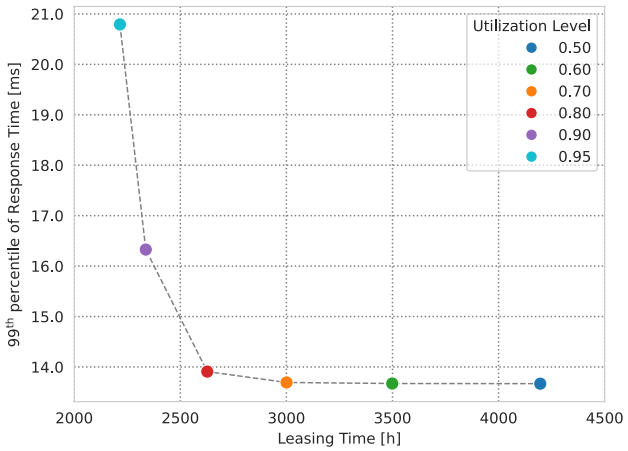

Fig. 10 99th percentile of the response time distribution as a function of the VM leasing time for different values of the desired utilization level

differences becoming larger and larger and reaching 2.6 and 7.1 ms, that is, 19% and 51.8% larger.

To further analyze the effects of the different utilization levels, we plot in Figure 10 the 99th percentile of the response time distribution as a function of the VM leasing time. The figure confirms that resource over-provisioning has positive effects in terms of response time even though at the expense of an increased leasing time. On the contrary, opposite effects can be detected in case of high utilization levels. We also remark that the utilization level equal to 0.8 represents a good tradeoff between request performance and VM "cost". In fact, it corresponds to the knee of the curve.

In summary, the extensive experiments showcased the benefits and robustness of our auto-scaling policy which is able to adapt in a timely manner to changes of the workload conditions. Moreover, its low overhead makes it a lightweight solution particularly suitable in production environments.

6 Conclusion

Auto-scaling solutions are of paramount importance to fully exploit the elasticity offered by cloud environments. In fact, allocating the "right" amount of resources for the incoming workload has significant benefits in terms of performance and monetary cost.

In this paper we proposed a lightweight, flexible framework for scaling cloud resources in a timely manner. Notably, to anticipate workload changes, the framework predicts the future workloads and the amount of resources necessary to cope with these workloads and ensures at the same time the desired overall resource utilization level.

We remark that our policy promptly triggers scale out actions, whereas it is rather conservative for scale in actions to avoid performance degradation due to premature resource deallocations.

To evaluate the proposed auto-scaling framework, we developed a simulation environment based on the CloudSim toolkit. Notably, the extensions mainly refer to an auto-scaler component that implements the auto-scaling policies and a monitoring component responsible of collecting measurements during the simulation.

Results showcased the effectiveness and robustness of the proposed framework which properly adjusts the allocated resources to workload patterns, thus preventing overload conditions. In particular, the experimental results have clearly shown that our policy outperforms the baseline policy in terms of total leasing time without affecting the request response times. The experiments have also demonstrated that low utilization levels increase leasing times without providing any significant benefit to the response time.

As future research directions, we will investigate the impact of VMs characterized by uncertain performance. In fact, the use of virtualization technologies and the co-location of heterogeneous workloads often lead to contention and performance degradation. To this end, we will model VM characteristics as random variables and modify the framework and the simulation environment accordingly.

We will also extend our auto-scaling framework to cope with the peculiarities of container-based platforms and compare its performance with existing auto-scaling solutions.

Author Contributions Authors contributed equally to this work.

Funding Open access funding provided by Università degli Studi di Pavia within the CRUI-CARE Agreement. This work was supported by the European Union - Next Generation EU - Mission 4, Component 1 (Master CUP: B53D23013090006, CUP: F53D23004300006)

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Smith, D.: Cloud Computing in 2028: From Technology to Business Necessity. Technical report, Gartner Research (2023)
2. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. J. Grid Comput. **12**(4), 559–592 (2014)
3. Qu, C., Calheiros, R.N., Buyya, R.: Auto-scaling Web Applications in Clouds: A Taxonomy and Survey. ACM Comput. Surv. **51**(4), 1–33 (2018)
4. Jacob, B., Lanyon-Hogg, R., Nadgir, D.K., Yassin, A.F.: A practical guide to the IBM Autonomic Computing toolkit. Technical report, IBM Redbooks (2004)
5. Gilly, K., Alcaraz, S., Juiz, C., Puigjaner, R.: Analysis of burstiness monitoring and detection in an adaptive Web system. Comput. Netw. **53**(5), 668–679 (2009)
6. Gilly, K., Juiz, C., Thomas, N., Puigjaner, R.: Adaptive admission control algorithm in a QoS-aware Web system. Inf. Sci. **199**, 58–77 (2012)
7. Chen, T., Bahsoon, R., Yao, X.: A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems. ACM Computing Surveys **51**(3), 1–40 (2018)
8. Verma, S., Bala, A.: Auto-scaling techniques for IoT-based cloud applications: a review. Clust. Comput. **24**, 2425–2459 (2021)
9. Radhika, E.G., Sudha Sadasivam, G.: A review on prediction based autoscaling techniques for heterogeneous applications in cloud environment. Materials Today: Proceedings **45**, 2793–2800 (2021)
10. Catillo, M., Villano, U., Rak, M.: A survey on auto-scaling: how to exploit cloud elasticity. Int. J. Grid Util. Comput. **14**(1), 37–50 (2023)
11. Iqbal, W., Dailey, M.N., Carrera, D.: Low Cost Quality Aware Multi-tier Application Hosting on the Amazon Cloud. In: Proc. of the Int. Conf. on Future Internet of Things and Cloud, pp. 202–209 (2014)
12. Augustyn, D.A., Warchal, L.: Metrics-based auto scaling module for Amazon Web Services cloud platform. In: Kozielski, S., Mrozek, D., Kasprowski, P., Malysiak-Mrozek, B., Kostrzewa, D. (eds.) Beyond Databases, Architectures, and Structures. Communications in Computer and Information Science, vol. 716, pp. 42–52 (2017)
13. Calzarossa, M.C., Massari, L., Tessera, D.: Evaluation of cloud autoscaling strategies under different incoming workload patterns. Concurrency and Computation: Practice and Experience **32**(17), e5667 (2020)
14. Liu, B., Buyya, R., Toosi, A.: A Fuzzy-Based Auto-scaler for Web Applications in Cloud Computing Environments. In: Pahl, C., Vukovic, M., Yin, J., Yu, Q. (eds.) Service-Oriented Computing - ICSOC 2018. Lecture Notes in Computer Science, vol. 11236, pp. 797–811 (2018)
15. Netto, M.A.S., Cardonha, C.H., Cunha, R.L.F., de Assunção, M.D.: Evaluating Auto-scaling Strategies for Cloud Computing Environments. In: Proc. of the 22nd Int. Symp. on Modeling, Analysis & Simulation of Computer and Telecommunications Systems - MASCOTS, pp. 187–196 (2014)
16. Gandhi, A., Harchol-Balter, M., Raghunathan, R., Kozuch, A.: AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. ACM Transactions on Computer Systems **30**(4), 14 (2012)
17. Gandhi, A., Dube, P., Karve, A., Kochut, A., Zhang, L.: Adaptive, Model-driven Autoscaling for Cloud Applications. In: Proc. of the 11th Int. Conf. on Autonomic Computing - ICAC'14, pp. 57–64 (2014)
18. Gong, Z., Gu, X., Wilkes, J.: PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In: Proc. of the Int. Conf. on Network and Service Management, pp. 9–16 (2010)
19. Roy, N., Dubey, A., Gokhale, A.: Efficient autoscaling in the cloud using predictive models for workload forecasting. In: Proc. of the IEEE 4th Int. Conf. on Cloud Computing - CLOUD'11, pp. 500–507 (2011)
20. Nikravesh, A.Y., Ajila, S.A., Lung, C.-H.: An autonomic prediction suite for cloud resource provisioning. Journal of Cloud Computing: Advances, Systems and Applications **6**(3), (2017)
21. Gandhi, A., Dube, P., Karve, A., Kochut, A., Zhang, L.: Model-driven optimal resource scaling in cloud. Software & Systems Modeling **17**, 509–526 (2018)
22. Qian, H., Wen, Q., Sun, L., Gu, J., Niu, Q., Tang, Z.: Robustscaler: QoS-aware autoscaling for complex workloads. In: Proc. of the IEEE 38th Int. Conf. on Data Engineering - ICDE, pp. 2762–2775 (2022)
23. Chouliaras, S., Sotiriadis, S.: An adaptive auto-scaling framework for cloud resource provisioning. Futur. Gener. Comput. Syst. **148**, 173–183 (2023)
24. Hang, H., Tang, X., Sun, J., Bao, L., Lo, D., Wang, H.: Robust auto-scaling with probabilistic workload forecasting for cloud databases. In: Proc. of the IEEE 40th Int. Conf. on Data Engineering - ICDE, pp. 4016–4029 (2024)
25. Calzarossa, M.C., Della Vedova, M.L., Massari, L., Petcu, D., Tabash, M.I.M., Tessera, D.: Workloads in the Clouds. In: Fiondella, L., Puliafito, A. (eds.) Principles of Performance and Reliability Modeling and Evaluation. Springer Series in Reliability Engineering, pp. 525–550 (2016)
26. Calzarossa, M.C., Massari, L., Tessera, D.: Workload characterization: A survey revisited. ACM Computing Surveys **48**(3), 1–43 (2016)
27. Kumar, J., Singh, A.K.: Workload prediction in cloud using artificial neural network and adaptive differential evolution. Futur. Gener. Comput. Syst. **81**, 41–52 (2018)
28. Calzarossa, M.C., Della Vedova, M.L., Massari, L., Nebbione, G., Tessera, D.: Modeling and predicting dynamics of heterogeneous workloads for cloud environments. In: Proc. of the IEEE Symposium on Computers and Communications - ISCC (2019)
29. Masdari, M., Khoshnevis, A.: A survey and classification of the workload forecasting methods in cloud computing. Clust. Comput **23**(4), 2399–2424 (2020)
30. Fernandez, H., Pierre, G., Kielmann, T.: Autoscaling Web Applications in Heterogeneous Cloud Infrastructures. In: Proc. of the IEEE Int. Conf. on Cloud Engineering - IC2E, pp. 195–204 (2014)
31. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Futur. Gener. Comput. Syst. **27**(6), 871–879 (2011)
32. Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: Proc. of the IEEE Network Operations and Management Symposium, pp. 204–212 (2012)
33. Rampérez, V., Soriano, J., Lizcano, D., Lara, J.A.: FLAS: A combination of proactive and reactive auto-scaling architecture for distributed services. Futur. Gener. Comput. Syst. **118**, 56–72 (2021)
34. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., Wood, T.: Agile dynamic provisioning of multi-tier internet applications. ACM Trans. Auton. Adapt. Syst. **3**(1), 1–39 (2008)
35. Bouabdallah, R., Lajmi, S., Ghedira, K.: Use of Reactive and Proactive Elasticity to Adjust Resources Provisioning in the Cloud Provider. Proc. of the IEEE 18th Int. Conf. on High Performance Computing and Communications; IEEE 14th Int. Conf. on Smart City; IEEE 2nd Int. Conf. on Data Science and Systems - HPCC/SmartCity/DSS, pp. 1155–1162 (2016)

36. J.V., B.B., Dharma, D.: HAS: Hybrid auto-scaler for resource scaling in cloud environment. Journal of Parallel and Distributed Computing **120**, 1–15 (2018)
37. Abdullah, M., Iqbal, W., Erradi, A., Bukhari, F.: Learning Predictive Autoscaling Policies for Cloud-Hosted Microservices Using Trace-Driven Modeling. In: Proc. of the IEEE Int. Conf. on Cloud Computing Technology and Science - CloudCom, pp. 119–126 (2019)
38. Singh, P., Kaur, A., Gupta, P., Gill, S.S., Jyoti, K.: RHAS: robust hybrid auto-scaling for web applications in cloud computing. Clust. Comput **24**(2), 717–737 (2021)
39. Joshi, N.S., Raghuwanshi, R., Agarwal, Y.M., Annappa, B., Sachin, D.N.: ARIMA-PID: container auto scaling based on predictive analysis and control theory. Multimedia Tools and Applications **83**(9), 26369–26386 (2024)
40. Zou, D., Lu, W., Zhu, Z., Lu, X., Zhou, J., Wang, X., Liu, K., Wang, K., Sun, R., Wang, H.: Optscaler: A collaborative framework for robust autoscaling in the cloud. Proceedings of the VLDB Endowment **17**(12), 4090–4103 (2024)
41. Amazon Web Services, Autoscaling Documentation. https://docs.aws.amazon.com/autoscaling/ [Accessed: April 30th, 2025]
42. Microsoft Azure, Autoscale. https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview [Accessed: April 30th, 2025]
43. Google Cloud Platform, Autoscale to maintain a metric at a target value. https://cloud.google.com/compute/docs/autoscaler/ [Accessed: April 30th, 2025]
44. Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Englewood Cliffs, New Jersey (1984)
45. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1), 23–50 (2011)
46. Juiz, C., Capo, B., Bermejo, B., Fernández-Montes, A., Fernández-Cerero, D.: A Case Study of Transactional Workload Running in Virtual Machines: The Performance Evaluation of a Flight Seats Availability Service. IEEE Access **11**, 81600–81612 (2023)