Contents lists available at ScienceDirect

Future Generation Computer Systems

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

An adaptive auto-scaling framework for cloud resource provisioning

Spyridon Chouliaras, Stelios Sotiriadis [*]

Department of Computer Science and Information Systems, Birkbeck, University of London, United Kingdom

A R T I C L E I N F O

Article history:
Received 2 September 2022
Received in revised form 13 May 2023
Accepted 20 May 2023
Available online 27 May 2023

Keywords:
Auto-scaling
Resource provisioning
Cloud computing
Convolutional neural networks
K-means

A B S T R A C T

Cloud computing emerged as a technology that offers scalable access to computing resources in conjunction with low maintenance costs. In this domain, cloud users utilize virtualized resources to benefit from on-demand and long-term pricing strategies. Although the latter consists of a more cost-efficient solution, it requires accurate estimations of future workload demands, which is a challenging task. Furthermore, clouds offer threshold-based auto-scaling rules that need to be manually controlled by the users according to application requirements. Still, tuning scaling parameters is not trivial, since it is mainly based on static scaling rules that may lead to unreasonable costs and quality of service violations. In this work we introduce ADA-RP, an adaptive auto-scaling framework for reliable resource provisioning in the cloud. ADA-RP uses historical time series data for training K-means and convolutional neural networks (CNN) to categorize future workload demands as High, Medium or Low based on CPU utilization. We auto-scale cloud resources in real-time based on the predicted workload demand to reduce costs and improve application performance. The experimental analysis is based on TPC-C runs on MySQL containers deployed on the Google Cloud Platform. Experimental results are prosperous, demonstrating the ability of ADA-RP (i) to reduce MySQL deployment costs by 48% in a single-tenant environment, and (ii) to double the executed queries per second in a multi-tenant environment considering user's budget requirements.

1. Introduction

Cloud computing is an emerging technology that presents new business opportunities due to its flexibility and on-demand availability. Fundamentally, it provides an infrastructure where various services and applications are available to users through the public Internet [1]. Cloud providers offer three service models namely as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In more detail, IaaS model includes virtualized resources in the form of Virtual Machine (VM) instances, while PaaS and SaaS offer higher levels of abstraction in terms of specific solution stacks and application software suites [2].

In this domain, cloud providers offer scalability and elasticity as key aspects of cloud computing systems. Cloud scalability refers to the ability of a system to increase or decrease resources to accommodate larger loads while cloud elasticity refers to the ability of a system to scale with loads dynamically [3]. Furthermore, cloud providers introduce flexible pricing models including on-demand plans that enable cloud users to pay just for the services they need and for the time they use them [4]. However, on-demand charging schemes are often more expensive than reserved plans where users get discount rates on reserved resources in specific availability zones [5]. Having said that, cloud users could benefit from long term contracts as long as they can produce accurate estimations of future workload demand. The latter, enables cloud users to launch a number of low-cost reserved instances to ensure high performance under heavy workloads. However, resource provisioning of cloud applications is not a trivial task. Often, the over-provisioning problem can occur where cloud users reserve more resources than the actual workload demand [6], leading to unnecessary costs. On the other hand, under-provisioning problem can occur when cloud users underestimate workload tasks and allocate inadequate amount of resources that often lead to Quality of Service (QoS) violations.

In this work, we present an adaptive auto-scaling framework for resource provisioning on the cloud, based on workload characterization and prediction. The proposed framework resolves the over-utilization and under-utilization problems by creating a scaling plan that automatically allocates cloud resources based on workload demand. To support our method, we deploy Linux containers in Google Cloud Platform (GCP) and we collect cloud metrics from various workload executions to form time series data. Then, we train K-means algorithm to cluster different workloads into a High, Medium or Low demand tasks based on their CPU utilization levels. Ergo, we extract three representative sequences of High, Medium and Low demand tasks by taking the average points of all sequences inside each cluster. Afterwards,

[*] Corresponding author.
E-mail address: s.sotiriadis@bbk.ac.uk (S. Sotiriadis).

the main application is being deployed and monitored under realistic workload executions to collect resource usage metrics. These are being used as an input to train the CNN model to predict future workload demand. The predicted sequence is being segmented and each segment inherits the label of the closest representative sequence based on the dynamic time warping algorithm. Lastly, based on the sequence labeling, ADA-RP generates a scaling plan that enables real-time and adaptive auto-scaling in cloud computing environments.

To support the proposed scaling method, we predict the demand of future events based on the CPU utilization metric. Thus, we introduce the ADA-RP (adaptive auto-scaling for resource provisioning) framework to avoid over-utilized and under-utilized resources that may lead to unnecessary costs and QoS violations. The proposed framework enables Service Level Agreement compliance for resource provisioning in IaaS. We use the GCP pricing calculator [7] to calculate cloud costs and ensure that the proposed solution does not exceed budget limits.

Having said that, the contributions of this work include the following:

- We proposed a learning framework for adaptive auto-scaling in cloud environments. The framework allows cloud applications to automatically allocate virtualized resources to manage workload demands without unnecessary costs.
- We enabled cloud users to take advantage of low-cost reserved instances to ensure high performance under heavy tasks by introducing a hybrid learning approach based on workload characterization and prediction.
- We achieved proactive auto-scaling to determine cloud resources in advance and to enable real-time resource scaling without introducing booting delays.
- We provided users the ability to choose the scaling decision time frequency based on application functionality.
- We evaluated the proposed method for adaptive resource provisioning under two different experimental scenarios that achieved: (i) reduced total deployment costs in a single-tenant environment and (ii) improved application performance in a multi-tenant cloud environment under concurrent workload executions. We used real world systems to demonstrate our solutions including GCP as the cloud platform, MySQL Relational Database Management System (RDBMS) containers as the deployed system and TPC-C [8] as a real world online transaction processing (OLTP) benchmark.

The proposed system enhances the ability of cloud consumers to provision cloud resources in both single and multi-tenant environments while significantly reduces the cloud costs and improves application performance. The rest of this paper is organized as follows: Motivation experiments are presented in Section 2. The system model is described in Section 3. Section 4 presents performance evaluation of the proposed framework under both single-tenant and multi-tenant cloud architectures. Related works are discussed in Section 5. Finally, conclusions and future work are stated in Section 6.

2. Motivation experiment

In this work, we focus on solving the concept of managing over-utilized and under-utilized containerized environments that affect system performance and may lead to system failures, performance degradation and unreasonable costs.

Cloud providers promise virtual resources that meet cloud application demands. However, cloud workloads may vary in type and frequently cloud users cannot decide the appropriate amount

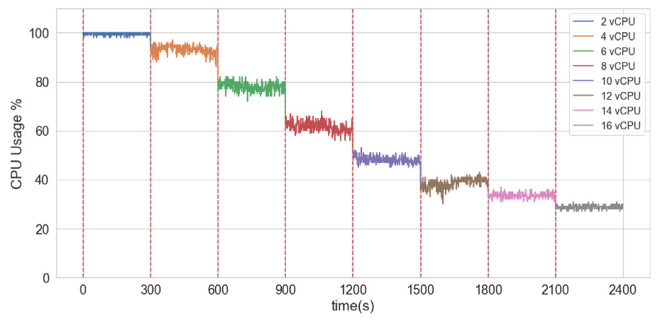

Fig. 1. CPU usage percentage based on vCPU cores allocation and TPC-C workload execution.

of resources in terms of gaining in performance and efficiency. The latter refers to resource allocation that meet application demand without introducing over-utilized or under-utilized resources while the former refers to the application performance such as throughput. Here, we focus on cloud workload characterization to identify application demand and adapt the cloud resources accordingly. Thus, we offer a framework that matches application performance to user requirements while at the same time reduces idle or over-utilized resources. Our motivation is based on Fig. 1 that experimentally demonstrates a real world system of a MySQL database server that is deployed in a Linux container in GCP. At the same time we execute the TPC-C workload, that is an online transaction processing (OLTP) benchmark and we use identical workload configurations for each run (10 warehouses and 10 connections). In addition, Prometheus[1] has been used as a monitoring engine to collect various metrics including CPU usage percentage. During the time of TPC-C execution, we utilized Linux cgroups to manage the amount of vCPU cores in the Linux container on-the-fly. As shown in Fig. 1, we increased the amount of allocated vCPU cores every 300 s time interval. We first allocated two vCPU cores in the container and we repeated this process for 4, 6, 8, 10, 12, 14 and 16 vCPU cores respectively. It has to be mentioned, that the container uses 64 GB of Memory throughout the whole duration of the experiment.

Fig. 1 shows that as we increase the amount of allocated vCPU cores in the container under the same workload execution, the CPU usage decrease proportionally. In more detail, the CPU usage remains above 70% for 2, 4 and 6 allocated vCPU cores while fluctuates between 70% and 45% for 8 and 10 allocated vCPU cores respectively. This decline continues as the CPU usage drops below 45% for 12, 14 and 16 allocated vCPU cores. This experimental scenario demonstrates that executing a realistic workload in a cloud application yields in both over-utilized and under-utilized resources. An over-utilized system might affect system reliability due to system failures and performance degradation while an under-utilized system introduces unreasonable cloud costs. We are motivated to create a framework to characterize workload demand as High, Medium and Low based on the CPU usage and provide scaling decisions to meet requirements for sufficient resource utilization.

Another challenge for system administrators is to ensure high performance in multi-tenant cloud environments. Following a multi-tenant architecture, tenants may share resources of a single instance to save costs. However, multi-tenant applications may experience performance degradation when tenants execute intensive workloads in parallel. To experimentally demonstrate a realistic scenario, we deployed a Virtual Machine with 12 vCPU cores and 64 GB Memory in GCP. Then, we deployed three Linux

[1] https://prometheus.io/

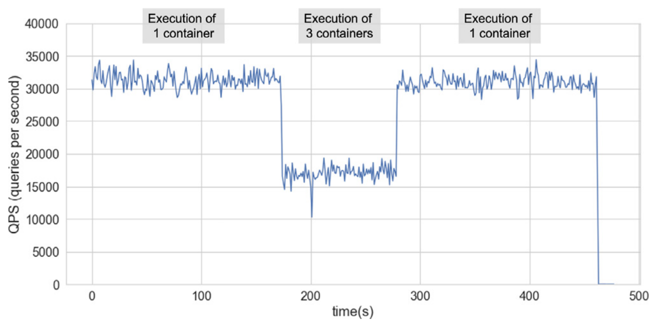

Fig. 2. Queries per second of MySQL container 1 over TPC-C workload with and without parallel workload executions.

containers to explore their performance under TPC-C workload executions while they equally share host resources. Fig. 2 illustrates the queries per second (QPS) that executed in the first MySQL container deployed in a multi-tenant environment.

As shown, more than 29,000 QPS are executed in the first container while the second and third containers remain idle (execution of 1 container). Then, between 181 and 280 s, we executed the same TPC-C workload to the second and third container (execution of 3 containers). This parallel workload executions negatively affected the performance of the first container, as the executed QPS drop below 18,000 in this time period. In the 281 s, we stopped TPC-C execution in the second and third containers (execution of 1 container). As a result, the QPS executed in the first container recovered to previous levels.

Motivated from the aforementioned scenarios, we propose a framework to ensure that resources are neither over-utilized or under-utilized and application performance meets the user Service Level Agreements (SLA). Consequently, we introduce a framework that: (i) automatically identifies the demand of a cloud application based on the CPU utilization, (ii) predicts future demand based on historical data, and (iii) creates a scaling plan to efficiently manage cloud resources for improving performance levels or reducing cloud costs.

3. Methodology

This section presents the ADA-RP framework, as an adaptive auto-scaler for cloud resource provisioning. The following section discusses (i) the ADA-RP framework, (ii) the system workflow, (iii) the dataset generation phase, (iv) the input parameters, (v) the algorithmic structure of the ADA-RP framework and (vi) the learning phase.

3.1. ADA-RP framework

ADA-RP is a scaling framework that automatically adjusts cloud application resources based on workload demand. The proposed framework utilizes the historical data from past workload executions to predict and characterize future application demands. To begin with, we deploy multiple cloud systems with different resource configurations (2 vCPU cores, 4 vCPU cores, 6 vCPU cores etc.,) that are being monitored under various workload scenarios. Such scenarios create different time series patterns in terms of the CPU utilization levels. The time-series data are being segmented into smaller sequences based on a time window length parameter, called *Wl* and used as an input to the K-means clustering algorithm that clusters each sequence to High, Medium and Low demand states. Then, we extract the representative sequence of each cluster by taking the average data points between all sequences that belong to similar clusters. Each representative sequence will be used to identify the state of future sequences.

Consequently, the main application is being deployed and monitored on the cloud. We execute TPC-C workload consecutively under the same configurations to generate repeatable patterns of the CPU utilization metric. Then, the CPU metric is being used as an input for the CNN model to predict the CPU utilization of a future workload execution based on a prediction step number parameter *Ps* that represents a time period. At that point, ADA-RP segments the predicted sequence into smaller sequences based on the time window length parameter *Wl*. The dynamic time warping algorithm is used to calculate the distance between each segment and the three representative wavelets of the High, Medium and Low demand clusters. Finally, each segment inherits the label from the closest representative sequence.

ADA-RP achieves on-demand scalability due to a proactive strategy that provisions cloud system resources in advance based on the CNN and K-means models. As for example, if the CNN model predicts the CPU utilization for the next 300 s ($Ps = 300$) and the time window length parameter *Wl* is 60 s, ADA-RP framework will segment the predictions into 5 sequences (Ps/Wl) where each sequence will be labeled as High, Medium or Low demand such as s_r= [$High, High, Medium, Medium, Medium$]. Then, ADA-RP will produce a scaling plan, called *Sp* for the next 300 s, such as $Sp = [Vc + Sv, Vc + Sv, Vc, Vc, Vc]$ where *Vc* represents the initial amount of vCPU cores and *Sv* represents the scaling parameter that specifies the amount of vCPU cores added or subtracted from the running system. This example suggests that the application should scale up the initial resources ($Vc + Sv, Vc + Sv$) for the first two segments [$High, High$] and return to the initial amount of vCPU cores (Vc, Vc, Vc) for the last three segments [$Medium, Medium, Medium$]. The proposed scaling algorithm is explained in more detail later in this section.

3.2. System workflow

In this section, we present the system workflow as shown in Fig. 3. First, we deploy a MySQL RDBMS in a Linux Container and we execute the TPC-C workload to demonstrate a real world case scenario. At the same time, Prometheus is being used as the monitoring framework that collects resource usage and application metrics used in this work. In step 1, the collected metrics are being pre-processed and stored as a historical dataset. The latter is being used in step 2 as an input training dataset for the CNN model to learn and forecast *Ps* number of future steps. In step 3, we generate the predicted sequence produced by the CNN model. In step 4, the predicted sequence is being segmented into Ps/Wl sequences based on the time window length parameter *Wl*. In step 5, we calculate the dynamic warping distance between each segment and each representative sequence produced by the pre-trained K- means clustering algorithm. Then, each segment will inherit the label (High, Medium or Low) of the closest representative sequence. In step 6, the scaling coordinator creates the scaling plan that consists all scaling decisions for each segment. In step 7, the deployment manager reads the scaling plan and check if it is valid in terms of resource availability and user's budget requirements. In step 8, the deployment manager schedules cloud system resources accordingly.

3.3. Dataset generation phase

Our solution is based on a data-driven approach to automatically scale cloud resources and ensure application performance. To support our method, we used TPC-C benchmark as a real world workload and Prometheus monitoring engine for data collection. As a result, two datasets have been generated: (i) a dataset for

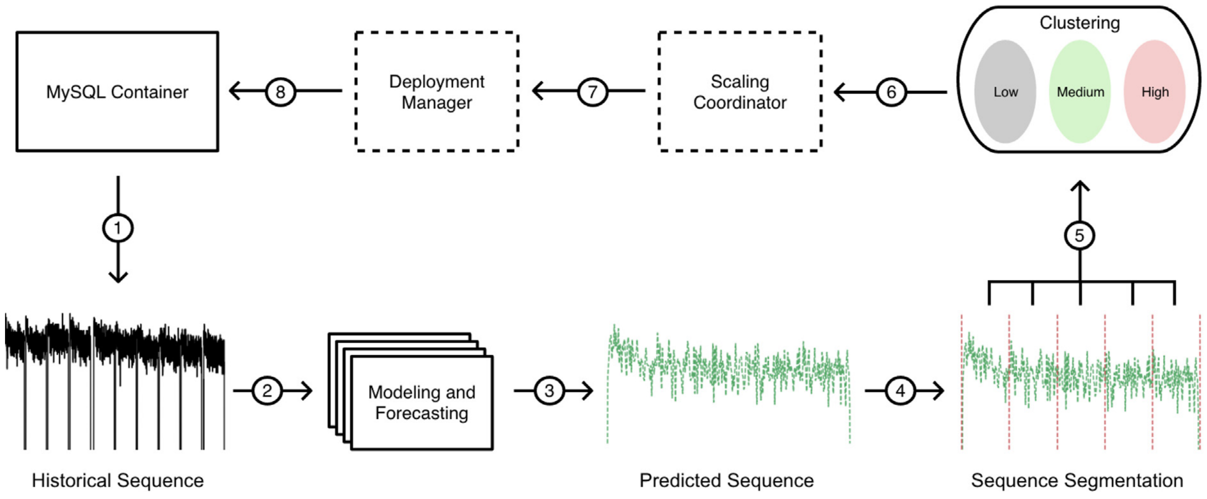

Fig. 3. System flow chart.

workload characterization using the clustering method and (ii) a dataset for workload prediction.

For the first dataset, we executed the TPC-C workload in a Linux MySQL container deployed on the GCP with different configurations. In more detail, we took the Cartesian product of two sets, called C and W to set all ordered pairs (c, w) where c is an element of C and w is an element of W. Thus, we give the following notation:

$$\mathrm{C}\times\mathrm{W} = \{(c, w)|c\in\mathrm{C} \text{ and } w\in\mathrm{W}\} \quad (1)$$

Set C = {2, 4, 6, 8, 10, 12, 14, 16} consists a 8-elements set where each element defines the vCPU cores allocated to a Linux container. Set W = {1, 5, 10, 15, 20} consists a 5-elements set where each element defines the number of Warehouses used in the TPC-C configurations. In total, 40 configurations have been used to generate unique CPU patterns while Prometheus monitoring engine collected the CPU utilization metrics. It has to be mentioned, that each run consists an average of 10 runs for ensuring reproducible data. Thus, we run 40 × 10 experiments, for 300 s each, and we average the results to obtain 40 runs, one for each configuration. In this work, ADA-RP makes scaling decisions every minute, thus each run of 300 s, is being segmented into five 60 s segments. As a result, we form a univariate time series dataset of 200 sequences that used to train the K-means clustering algorithm. The latter, categorizes each unique sequence into High, Medium or Low demand clusters based on the CPU utilization levels.

To support the workload prediction task, we deployed and monitored MySQL Linux containers in GCP so as to generate the second dataset. Then, we executed TPC-C workload consecutively to demonstrate real world experimental scenarios. As a result, a historical univariate time series has been formed for each scenario with a single column of CPU utilization and a timestep for each value as an implicit variable. The historical sequence has been pre-processed and used as an input for the CNN model to predict future events.

3.4. Input parameters

The proposed framework considers the following five parameters to initiate a scaling decision: (i) the prediction step number *Ps* parameter, (ii) initial vCPU cores *Vc*, (iii) the scale parameter *Sv*, (iv) the time window length parameter *Wl*, and (v) the Budget limit *Bl* parameter.

The prediction step number *Ps* defines the length of the predicted sequence produced by the CNN model. The initial vCPU cores parameter *Vc* defines the number of vCPU cores of the running system before ADA-RP scaling decisions and is being extracted automatically from Linux Unix shell.

The scale parameter *Sv* is a positive integer number that specifies the amount of vCPU cores added or subtracted from a running system. As for example, if the scale parameter *Sv* is equal to x and the current vCPU core number is *Vc*, ADA-RP will add or subtract x vCPU cores to the running application resulting in *Vc* + x or *Vc* - x allocated vCPU cores in a scale up and a scale down scenario respectively. To avoid negative vCPU core values we require that *Vc* - *Sv* > 0. On the one hand, increasing the value of *Sv* parameter implies that the target application will reach faster a Medium demand state either from a High demand state (over-utilized resources) or from a Low demand state (under-utilized resources). However, significantly high values may disrupt the application by passing from High to Low without reaching the desirable Medium demand state and vice versa. On the other hand, assigning low values to *Sv* parameter leads to slower adaptation to the Medium state but with higher confidence that the target application will reach the Medium demand state.

The time window length *Wl* is a positive integer number that defines the time, measured in seconds, before ADA-RP makes a new scaling decision. Let *Wl* be the time window length and *Ps* the CNN prediction step number, then ADA-RP takes *Ps*/*Wl* scaling decisions in *Ps* seconds while we require that *Ps* is divisible by *Wl* (*Wl* | *Ps*). On the one hand, increasing the value of *Wl* parameter lead to a decreased flexibility of the workload adaptation since ADA-RP will produce less decisions between *Ps* seconds. On the other hand, low values of *Wl* parameter lead to a higher flexibility since ADA-RP will adapt more precisely to workload demands. In this work *Wl* configuration is *Wl* = 60 meaning that ADA-RP takes scaling decisions for the cloud application every minute. Finally, the Budget limit parameter *Bl* defines the maximum cloud cost that is a user requirement constraint used by the deployment manager.

3.5. Algorithmic structure of the ADA-RP framework

ADA-RP consists of a hybrid machine learning system that enables adaptive resource provisioning on the cloud. Algorithm 1 illustrates the input and output data as well as the step-wise procedure of ADA-RP framework. As previously discussed, ADA-RP uses a time series dataset for workload characterization namely as *Tw*, a time series dataset for workload prediction namely as *Tp*. In detail, *Tw* used for training K-means algorithm and *Tp* is used as a time series historical data to training CNN to produce time series predictions. Furthermore, the input parameters namely as prediction step number *Ps*, initial vCPU cores *Vc*, scale parameter *Sv*, time window length *Wl* and the budget limit *Bl*.

We start ADA-RP by initializing an empty array *Sp* that will store the scaling plan. Then, the CNN model is trained on time series data collected by monitoring the target application.

Algorithm 1 Pseudocode for ADA-RP

```
Input
    $Tw$ Time series for workload characterization
    $Tp$ Time series for workload prediction
    $Ps$ Prediction step number
    $Wl$ Time window length
    $Vc$ Initial vCPU cores
    $Bl$ Budget limit
    $Sv$ Scale parameter
Output
    $Dp$ Deployment Plan
1: procedure ADA-RP
2:     Initialize an empty array $Sp$
3:     function CNN($Tp, Ps$)
4:         $CNN\_train()$
5:         $CNN\_predict()$
6:         return $Predicted\_Sequence$
7:     end function
8:     function K-means($Tw$)
9:         $K-means\_train()$
10:        $DTW\_Barycenter\_Averaging()$
11:        return $R$                         ▷ Representative Sequences
12:    end function
13:    function Segment($Predicted\_Sequence, Wl$)
14:        return $S$                         ▷ Segmented Sequences
15:    end function
16:    function DTW($R, S$)
17:        return $Ls$                        ▷ Labeled Sequences
18:    end function
19:    function Scale_Plan($Ls, Vc, Sv$)
20:        for each $label$ in $Ls$ do
21:            if $label == High$ then
22:                $Sp.append(Vc + Sv)$
23:            else if $label == Low$ then
24:                $Sp.append(Vc - Sv)$
25:            else
26:                $Sp.append(Vc)$
27:            end if
28:        end for
29:        return $Sp$
30:    end function
31:    function DEPLOYMENT_MANAGER($Sp, Bl$)
32:        $Deploy(Sp, Bl)$                    ▷ Budget-aware Deployment
33:        return $Dp$
34:    end function
35: end procedure
```

During the training process the prediction step number *Ps* is being considered to estimate the future workload demand, that is the *Predicted Sequence*. Additionally, the K-means model is trained on the time series data for workload characterization to cluster each sequence into High, Medium and Low demand clusters. Consequently, three cluster representative sequences are being created by using the *DTW Barycenter Averaging* (DBA) that is a method to calculate the average sequences explained later in this section. Given $k = 3$ number of clusters, let R be a set of three representative sequences as follows:

$$R = \{H, M, U\} \tag{2}$$

where $H = \langle h_1, h_2, \ldots, h_l \rangle$ is the representative sequence for High demand cluster, $M = \langle m_1, m_2, \ldots, m_l \rangle$ is the representative sequence for Medium demand cluster and $U = \langle u_1, u_2, \ldots, u_l \rangle$ is the representative sequence for Low demand cluster respectively, while l is the length of each sequence. It has to be mentioned that each representative sequence has the same length with the time window length where $Wl = l$ seconds.

The *Predicted Sequence* sequence is being segmented into Ps/Wl sequences based on the time window length parameter Wl that defines the length of each segment, therefore the frequency of decision making. Given a predicted sequence of $Ps = p$ time steps and the time window length parameter set to $Wl = l$ seconds, ADA-RP will produce $s = Ps/Wl$ sequence segments as follows:

$$S = \{S_1, S_2, \ldots, S_s\} \tag{3}$$

where S_s is the s-th segmented sequence of length l. Then, we calculate the distance between each segment in the segmented sequences S and the representative sequences R produced by the K-means model.

Ergo, each segment sequence inherits the label from the closest representative sequence. As a result an array of labels is constructed as follows:

$$Ls = (y_1, y_2, \ldots, y_s) | y \in A \tag{4}$$

where the label y_i is an element belonging to a finite set of classes A = {*High*, *Medium*, *Low*} indicate the state of the running application in terms of CPU utilization level. Then, a scaling plan is being created based on the labeled array *Ls*, the initial vCPU cores *Vc* and the scale parameter *Sv*. Finally, the deployment manager will initiate the scaling decisions based on user's cost limit requirements.

3.6. Learning phase

In this work, a hybrid learning framework has been proposed to enable adaptive auto-scaling on cloud applications. As mentioned in the dataset generation phase (Section 3.3), a dataset has been generated under different system and workload configurations and used to train the clustering algorithm. In the past, a variety of algorithms have been used for clustering time-series including fuzzy c-means [9], Gaussian Mixture Models [10] and Hierarchical Agglomerative clustering [11]. However, this work uses the K-means algorithm which has been widely used across a wide range of research fields for clustering time-series data [12–14].

K-means clusters time series samples into k predefined number of groups by trying to minimize the Within Group Sum of Squares (WGSS) also called inertia, a measure for cluster coherence. The K-means randomly assigns each sample to a cluster and computes the average cluster sequence. Then, each sample is being re-assigned to the closest average cluster sequence and the average sequences are being re-calculated. The same process is being repeated until a decrease of the objective function is lower than a tolerance value.

In this domain, various strategies have been proposed for averaging a set of sequences. Local averaging techniques, including the NonLinear Alignment and Averaging Filters (NLAAF) [15] and Prioritized Shape Averaging (PSA) [16], have been proposed to compute the average sequence over a set of sequences. Such techniques compute the mean of N sequences by taking the pairwise averaging. However, pairwise averaging is quite sensitive to order, thus repeating the averaging process may significantly alter the quality of the result. For that reason, authors in [17] proposed the *DTWBarycenterAveraging* (DBA), a global averaging method, that iteratively refines the initial average sequence to minimize its squared Dynamic Time Warping (DTW) distance to averaged sequences. For each refinement DBA works in two steps: (i) computes the DTW distance between each sequence and the temporary average sequence and (ii) updates each coordinate of the average sequence based on the contribution of one or more coordinates of each sequence.

In step 1, the DTW distance between the average sequence and an individual sequence involves the searching sequence C at iteration i for instances on sequence T as:

$$C = \langle C_1, C_2, C_3, \dots, C_l \rangle \tag{5}$$

$$T = \langle T_1, T_2, T_3, \dots, T_l \rangle \tag{6}$$

where C is the average sequence, T is a sequence associated to it and l is the length of sequences. It has to be mentioned, that DTW can align sequences of different lengths, however in this work the average sequence and the segment sequences are of the same length.

As a result, the sequences C and T are used to form a l-by-l square matrix where each point (i, j) corresponds to an alignment between elements C_i and T_j. The cumulative distance for each point in the matrix is being calculated using dynamic programming to evaluate the following recurrence:

$$\gamma(i,j) = \delta(i,j) + min[\gamma(i-1,j), \gamma(i-1,j-1), \gamma(i,j-1)] \tag{7}$$

that is the sum of the distance between two elements and the minimum of the cumulative distances of the adjacent cells. In DBA, the goal is to minimize the sum of squared DTW distances between the average sequence and the set of sequences by refining the average sequence iteratively. Thus, in step 2, DBA updates each of the average coordinates as the barycenter of coordinates associated to it during the first step. Given a set of sequences $S = \{S_1, \dots, S_N\}$ to be averaged, DBA starts from the initial average sequence $C = \langle C_1, \dots, C_l \rangle$ and iteratively refines the average sequence to $C' = \langle C_1', \dots, C_l' \rangle$, that is the updated average sequence at iteration $i + 1$, where l is the length of sequences and N is the number of sequences.

As previously mentioned, ADA-RP uses a pro-active approach to reduce cloud costs and system inefficiencies based on CNN future predictions. The CNN model has been widely used in the past for predicting time series [18,19]. In our work the CNN learns from the historical time series in order to estimate future workload demand. Time series consist of a single dimension, that is the time dimension, thus can be defined as one dimensional vector. As a result, CNN uses one dimensional convolutions to process the input time series.

During the learning phase, several hyperparameters considered to fine tune the CNN model. Table 1 shows the best values of the hyperparameters that obtained after conducting various experiments.

The CNN model contains an input layer, two convolutional layers, two pooling layers, a fully connected layer and an output layer. The Convolutional layers consist of 64 filters and a kernel size of two. The max pooling operation is performed after each convolutional layer to downsample the input representation by taking the maximum value over a spatial window of size two. Then, we perform flattening to convert the convolved and pooled features into one dimensional array that is passed to a fully connected layer. Lastly, an output layer generates a predicted sequence of equal size with the input sequence. Rectified Linear Unit (Relu) [20] is applied after each convolution operation and Adam optimizer used for first-order gradient-based optimization based on adaptive estimates of lower-order moments [21]. The learning rate set to 0.0001 and the model is trained for 100 epochs with a batch size of 64. To evaluate the effectiveness of the proposed model, we use the Root Mean Squared Error (RMSE). The latter can be represented mathematically as follows:

Table 1
Optimal CNN configurations.

Parameter	Value
Convolutional layers	2
Convolutional layer filters	64
Pooling layers	2
Fully connected layer	1
Fully connected layer neurons	300
Optimizer	Adam
Activation function	Relu
Loss function	RMSE
Learning rate	0.0001
Number of epochs	100
Batch size	64

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{8}$$

where y_i and $\hat{y}_i$ are the observed and the predicted values, and n is the total number of observations. The train and the test RMSE values are being reported in the performance evaluation section.

4. Performance evaluation

This section presents the performance evaluation of ADA-RP framework based on TPC-C runs in MySQL, where each run refers to a single workload execution for a duration of 300 s. The experimental results are based on empirical decisions that better demonstrate the effectiveness of the proposed framework. In that context, we make the assumption that the amount of vCPU cores added or subtracted from the running system (Sv parameter) will affect the overall system performance due to CPU intensive activity on the part of the database server. Additionally, the QPS threshold is set to 10% ensuring that cost-effective scaling decisions will not lead to performance degradation. Lastly, the system uses a budget limit parameter set to 500$ per month to ensure that scaling decisions do not exceed user's cost requirements.

Having said that, K-means algorithm trained on a historical dataset to generate three representative sequences for High, Medium and Low demand clusters. These are being used to label future segmented sequences and allocate resources accordingly. It has to be mentioned that these segmented sequences refer to five 60 s segments (1 to 5) derived from a 300 s sequence. Two evaluation scenarios are presented as follows: (i) adaptive auto-scaling in a single-tenant architecture to avoid under-utilized resources and decrease cloud costs and (ii) adaptive auto-scaling in a multi-tenant architecture to avoid over-utilized resources and ensure high application performance.

4.1. K-means clustering for time series

As mentioned earlier, a historical dataset for workload characterization has been generated based on different system and application configurations. The latter, used to train the K-means algorithm with a predefined value of $k = 3$, that is the number of clusters. As shown in Fig. 4, the K-means algorithm clusters time series data into High, Medium and Low demand clusters based on their CPU utilization levels. Each time series consists of 60 timesteps over a 60 s period of time.

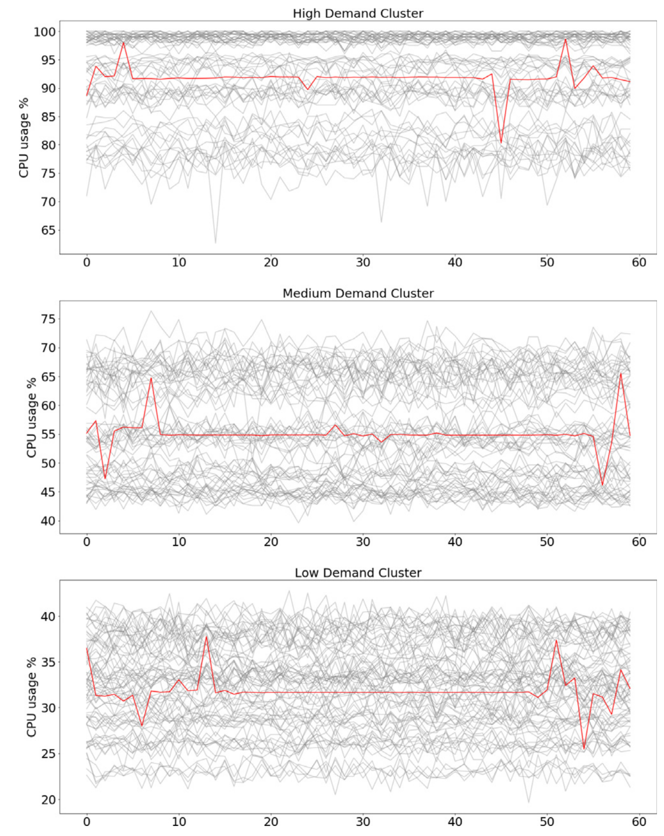

Fig. 4. K-means algorithm for clustering time series data into High, Medium and Low demand clusters over a 60 s period of time. The average sequence for each cluster is shown in red color.

The DBA averaging method has been used during the K- means learning algorithm that produced an intraclass inertia of 0.197. Fig. 4 shows the average cluster sequences in red color that used as representative sequences for High, Medium and Low demand clusters respectively.

The representative sequence for High demand cluster fluctuates between 81% and 98% suggesting over-utilized CPU resources. On the other hand, the representative sequence for the Medium demand cluster consists neither under-utilized nor over-utilized resources with values between 47% and 66% respectively. Finally, the representative sequence for the Low demand cluster takes values between 26% and 38% suggesting that the running system is under-utilized. ADA-RP enables an auto-scaling method for cloud resource provisioning. In this domain, K-means automatically characterize future workload demand based on CPU utilization levels without user intervention (e.g., threshold-based scaling rules).

4.2. Adaptive auto-scaling in single-tenant architecture

In this experiment, we present an experimental evaluation scenario to demonstrate the effectiveness of ADA-RP framework in a single-tenant cloud environment. The experimental platform includes a VM with 16 vCPU cores and 64 GB of Memory deployed in GCP. We deployed a MySQL Linux container and we used the Linux cgroups to limit and isolate CPU resources based on workload demands. Fig. 5 shows the CPU usage of MySQL container and the vCPU cores allocation while we execute the TPC-C workload. As shown in Fig. 5, the MySQL container has no limitation to CPU resources while accessing 16 vCPU cores (VM total) in the first 10 TPC-C runs. This results to under-utilized CPU

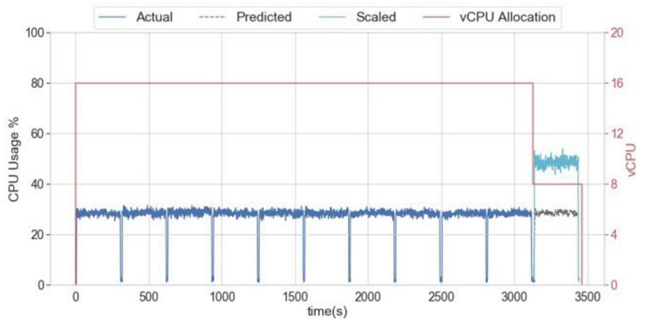

Fig. 5. CPU usage percentage, CPU usage predictions and virtual CPU allocation based on adaptive auto-scaling strategy for single tenant architecture.

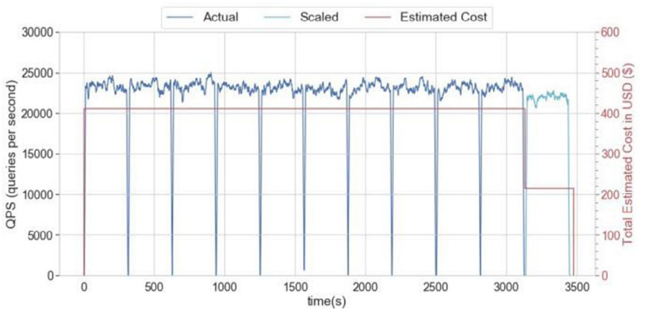

Fig. 6. QPS of MySQL container and total estimated costs based on adaptive auto-scaling strategy for single-tenant architecture.

resources of MySQL container since the CPU usage percentage fluctuates between 26% and 32% respectively.

To adapt to workload demands and reduce under-utilized resources, ADA-RP uses the historical data as an input to train the CNN model and produces CPU usage predictions of 300 s (gray dotted line). We show 10 (out of 20) training wavelets used to extract the predicted wavelet while we achieved an RMSE score of 0.0162 in the training data and 0.0181 in the testing data respectively. Consequently, the predicted sequence is being segmented into 60 s intervals ($Wl = 60$) and used as an input for the K-means algorithm that clusters each segment into a High, Medium or Low demand cluster. As a result, ADA-RP creates a scaling future plan for the next 300 seconds that contains all scaling decisions for each 60 s segment. As shown in Fig. 5, ADA-RP reduces the allocated vCPU cores to avoid under-utilized resources in the next TPC- C workload run. In more detail, each segment has been characterized as Low demand, thus, ADA-RP scales down in advance the MySQL container for the next 300 s (segments 1–5) from 16 to 8 vCPU cores respectively. The latter results to an increase in CPU usage percentage of MySQL container (light blue line) that fluctuates between 46% and 52% respectively (Medium demand).

Furthermore, ADA-RP extracts application performance metrics and calculates cloud costs to ensure high performance without unreasonable costs. Fig. 6 shows the QPS and the cloud costs of MySQL container at the same time interval as Fig. 5.

In the first 10 TPC-C runs, the MySQL container has access to 16 vCPU cores with a cost of 411.35$ per month while the executed QPS fluctuated between 21,000 and 24,500 respectively. However, in the last TPC-C run, ADA-RP reduces vCPU cores from 16 to 8 that resulted to a significant cloud cost drop to 215.67$ per month. The reduce of vCPU cores is based on the scale factor parameter $Sv = 8$ set by the user. Thus, ADA-RP reduces cloud costs by 48% with a decrease of 6.9% in executed QPS that fluctuated between 20,600 and 22,800. For these experiments,

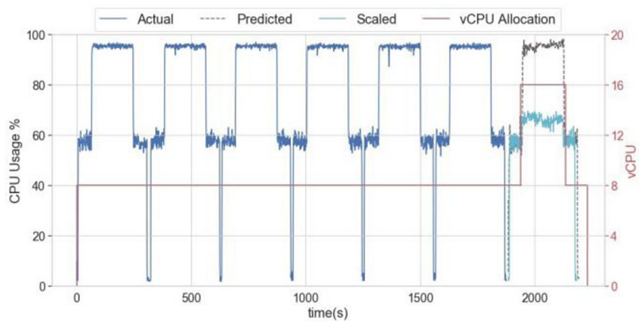

Fig. 7. CPU usage percentage, CPU usage predictions and virtual CPU allocation based on adaptive auto-scaling strategy for multi-tenant architecture.

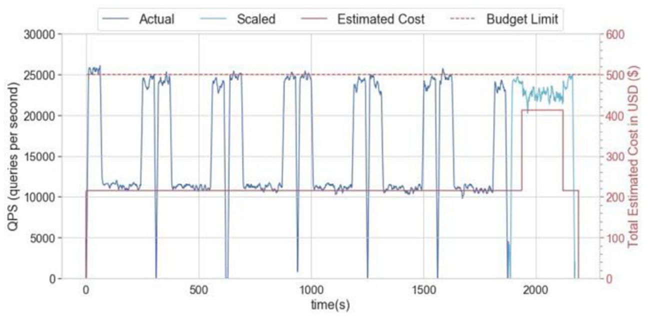

Fig. 8. QPS of MySQL container and total estimated cost based on adaptive auto-scaling strategy for multi-tenant architecture.

we set an acceptable reduction threshold to 10%, meaning that if the QPS drops below this threshold for 60 s, ADA-RP returns to the initial resource configuration. Here, we demonstrated the ability of ADA-RP framework to reduce under-utilized resources in a single-tenant cloud environment, by reducing the executed QPS. This tradeoff between cost and performance can be adjusted by the user, by introducing an acceptable performance reduction threshold. ADA-RP improves the efficiency of the containerized cloud application by decreasing both the CPU idle resources and the cloud costs within service level agreement.

4.3. Adaptive auto-scaling in multi-tenant architecture

In this experiment, we present the ability of ADA-RP framework to take automatic scaling decisions in a multi-tenant cloud environment. In such environments, tenants may share resources of a single instance to achieve economies. As a result, the system is often over-utilized and the containerized application may experience performance degradation. To demonstrate such a scenario, we deployed a VM with 16 vCPU cores and 64 GB of Memory in GCP. Furthermore, we deployed three MySQL Linux containers and through cgroups we limited the resources of each container to 8 vCPU cores. It has to be mentioned that all three containers have been isolated to use the same number of vCPU cores (1–8) while (9–16) vCPU cores remain unallocated. Fig. 7 illustrates the CPU usage and the vCPU cores allocation of *container*1 while we execute the TPC-C work-load to create repeatable workload patterns. In more detail, each TPC-C run consists of 300 s execution time for *container*1 and 180 s execution time for *container*2 and *container*3. As shown in Fig. 7, the CPU usage of *container*1 fluctuates between 54% and 62% for the first and last 60 seconds of each TPC-C run. In that period, *container*2 and *container*3 remain idle. Then, to demonstrate a multi-tenant environment where tenants require resources concurrently, for each TPC-C workload execution in *container*1, we additionally execute the TPC-C workload inside *container*2 and *container*3 between 61–240 s of each 300 s execution. This resulted to a sharp increase in the CPU usage percentage of *container*1 that fluctuates between 93% and 97% respectively.

Fig. 7 demonstrates the ability of ADA-RP to adapt to workload demand and avoid over-utilized resources. As shown in Fig. 7, the CNN model generates future CPU usage predictions of 300 s (gray dotted line) while achieving an RMSE score of 0.0352 in the training data and 0.0627 in the testing data. We show 6 (out of 20) training wavelets used to extract the predicted wavelet. The latter has been segmented into five 60 s intervals ($Wl = 60$) that used as an input for the K-means clustering algorithm. Ergo, ADA-RP takes decisions about container vCPU cores allocation every minute. Then, ADA-RP creates a scaling plan that includes scaling decisions for each segment to adjust the allocated resources in the container based on workload demand and system availability.

As shown in Fig. 7, in the 7th TPC-C execution, ADA-RP does not scale *container*1 between 1886–1945 and 2126–2185 s (segments 1 and 5) since the K-means clusters both segments as Medium demand. However, between 1946 and 2125 s (segments 2, 3 and 4), the K-means identifies the over-utilized CPU resources and clusters all three segments as High demand. Therefore, ADA-RP increases the allocated vCPU cores from 8 to 16 between 1946–2125 s (segments 2, 3 and 4) based on a scale factor parameter $Sv = 8$ set by the user. As a result, the state of *container*1 in that time period changed from High to Medium demand as the CPU usage decreased significantly and fluctuated between 58% and 66% respectively.

Additionally, Fig. 8 shows the QPS and the cloud costs of *container*1 at the same time interval with Fig. 7. On the one hand, the executed QPS in *container*1 fluctuates between 22,800 and 26,200 while *container*2 and *container*3 remain idle. On the other hand, the executed QPS in *container*1 sharply declined between 9800 and 12,200, since *container*2 and *container*3 executed the TPC-C workload in parallel. However, the decision of ADA-RP framework to increase the allocated CPU resources from 8 to 16 vCPU cores resulted to application performance recovery. As shown in Fig. 8 the executed QPS in *container*1 significantly increased and fluctuated between 20,400 and 23,500 (light blue line) during the parallel workload execution of the last run.

Fig. 8 shows that during the last TPC-C run, ADA-RP framework manages to scale up the CPU resources of MySQL *container*1 to avoid over-utilized resources and ensure high application performance. However, as illustrated in Fig. 8, additional CPU resources resulted to an increase in cloud costs from 215.67$ to 411.35$ respectively. To comply with user requirements, ADA-RP considers a budget limit parameter before decision making. In this experiment, the ADA-RP calculates the costs of the recommended configurations based on GCP pricing calculator and scales up *container*1 without exceeding the budget limit which is set by the user to 500$ per month.

5. Related work

Recent works suggested different auto-scaling techniques to facilitate resource provisioning in the cloud. In [22] authors suggested that most reviewed works can be fit in one or more of the following categories: (a) threshold-based rules, (b) reinforcement learning, (c) queuing theory, (d) control theory and (e) time series analysis.

Threshold-based auto-scaling rules enable cloud users to define an upper and a lower threshold based on one or more performance metrics such as CPU utilization and average response time. As a result, if the performance metric value is over the upper threshold or under the lower threshold then a scaling decision will be triggered. In [23] authors defined a threshold-rule, that is the average response of the application, to enable cost-efficient elasticity. In [24] authors introduced an elastic and scalable data streaming system where provisioning and decommissioning are triggered based on an upper and lower CPU utilization threshold. Threshold-based techniques promise lightweight and faster solutions, however setting the corresponding thresholds is not a trivial task since it requires a deep understanding of the running application.

Reinforcement learning techniques are suitable for automatic scaling decisions as they do not require a priori knowledge of the application performance model. In reinforcement learning an agent, that is the auto-scaler, reinforces scaling decisions that lead to high rewards. The reward system is based on application performance improvement such as increase application throughput or reduce response time. In [25,26] authors use a reinforcement learning algorithm known as Q-learning to determine optimal scaling policies. In [27] authors proposed a self-adaptive fuzzy logic controller that combines two reinforcement learning approaches: (i) Fuzzy SARSA learning and (ii) Fuzzy Q-learning. However, reinforcement learning algorithms introduce several problems including a long training period of time and poor performance results until an optimal policy is found.

Control Theory has been widely used for automated resource management in cloud environments. In [28] authors proposed multi-input, multi-output (MIMO) resource controller that detects CPU and disk I/O bottlenecks and allocates the right amount of virtualized resources to mitigate them. However, their approach does not control application adaptive parameters which can impact both the application benefit and execution time. In [29] authors introduce two adaptive hybrid controllers that use both active and proactive control for scaling cloud resources based on present and future demand respectively.

Queuing theory enables auto-scaling techniques by modeling applications and systems based on specific performance metrics such as the queue length or the average waiting time for requests. In [30] authors used queuing theory to formalize the problem of resource allocation in cloud computing environments.

They proposed optimal solutions considering various QoS parameters such as arrival rates and services resources. In [31] authors proposed a novel cloud resource auto-scaling scheme for web application providers. They predicted the number of requests by exploiting machine learning techniques and then they utilized queuing theory and multi objective optimization to discover the optimal number of VMs.

Time series analysis has been widely used to model and forecast future values of an application metric [32,33]. In this domain, time series analysis can be used as a proactive approach in the auto-scaling process, where a suitable scaling action could be planned based on the predicted values. The latter, enables the target system to scale in advance to adapt to workload demands without introducing booting delays. In [34] authors proposed an autonomic scaling decision model based on future resource utilization. They proposed Artificial Neural Networks to predict the CPU utilization of a running application to enable adaptive resource provisioning in the cloud and facilitate dynamic and proactive resource management. In [35] authors considered the number of user requests per time unit as the performance metric and utilized Support Vector Machine and Artificial Neural Networks as time-series prediction techniques.

In [36] authors suggested that IaaS cloud inevitably brings noticeable performance overhead to the running system due to the VM shared computing resources in datacenters. In that context, a variety of auto-scaling strategies used to adjust cloud resources while taking into account the computational overhead [37–40]. In [37] authors proposed MarVeLScaler, a multi-view deep learning model to capture real-time performance variance and automatically scale out the cloud cluster. MarVeLScaler consists of two models namely as Scale Estimator and Scale Controller. Scale Estimator estimates the required cluster size given a MapReduce job while Scale Controller adjusts cluster resources according to its real-time running status to guarantee the job finished on time. In [38] authors proposed Heifer, a Heterogeneity and interference aware VM provisioning framework for tenant applications. Heifer predicts the performance of MapReduce applications using the online-measured resource utilization of VM instances and capturing VM interference.

In [39] authors proposed Cocoa (COmputing in COntAiners), a container-based framework to implement group buying for cloud resources. In Cocoa, user jobs are grouped together and allocated either to newly created group buying deals or to existing deals with ideal resources with a price incentive. Additionally, the authors combined a static grouping strategy with a dynamic strategy to avoid the overhead of live migration when dealing with the strong dynamics of job arrivals and departures. In [40] authors introduced RACE, a Reliability-Aware server Consolidation strat-Egy to address when and how to perform energy-efficient server consolidation in a reliability-friendly and profitable way. They formulated the problem as a multi-objective optimization task where an improved genetic algorithm used to search the global optimal solution that unifies multiple constraints on performance SLAs, reliability factors and energy costs in a holistic manner.

In this work, we introduce a hybrid machine learning approach for auto-scaling containerized cloud applications. This work enables cloud vertical auto-scaling to increase or decrease the size of the containers based on future workload demand. We use Convolutional Neural Networks (CNNs) to model time series data and predict future values based on a historical time window. Then, the predicted values are being segmented and classified as High, Normal or Low demand based on K-means and Dynamic Time Warping algorithm. Our approach enables auto-scaling without predefined thresholds while at the same time introduces adaptation to workload changes.

6. Conclusions

In this work, we presented ADA-RP, an adaptive auto-scaling framework for cloud resource provisioning, based on workload characterization and prediction. ADA-RP uses the CPU utilization metric from previous workload executions while utilizes K-means and CNN models to develop a hybrid learning system for decision making. The framework provides adaptive auto-scaling to avoid over-utilized and under-utilized resources that may lead to unnecessary costs and QoS violations. We demonstrated an experimental analysis that reduces total deployment costs in a single-tenant cloud environment and improves application performance in a multi-tenant cloud environment under concurrent workload executions. The assumption is that the patterns are repeatable since TPC-C workload generated the same patterns for the selected specifications. However, repeatable workload patterns is a potential threat to validity for this study. In the future, we aim to explore reactive resource provisioning approaches to enhance the decision-making process under non-repeatable workload patterns. We will further generate and explore different datasets and systems, including NoSQL, to enhance our models and improve scaling decisions of the ADA-RP framework. Lastly, a future work includes the exploration of anomaly detection methods to ensure that scaling decisions are not prone to error due to abnormal system behavior.

CRediT authorship contribution statement

Spyridon Chouliaras: Conceptualization, Methodology, Formal analysis, Software, Data curation, Writing – original draft, Software, Visualisation. **Stelios Sotiriadis:** Validation, Investigation, Methodology, Investigation, Supervision, Writing – reviewing & editing, Project administration.

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.

Data availability

No data was used for the research described in the article.

References

[1] Stelios Sotiriadis, Nik Bessis, An inter-cloud bridge system for heterogeneous cloud platforms, Future Gener. Comput. Syst. 54 (2016) 180–194.
[2] Dario Bruneo, A stochastic model to investigate data center performance and qos in iaas cloud computing systems, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2013) 560–569.
[3] Stelios Sotiriadis, Nik Bessis, Cristiana Amza, Rajkumar Buyya, Vertical and horizontal elasticity for dynamic virtual machine reconfiguration, IEEE Trans. Serv. Comput. 99 (2016) 1.
[4] Robert L. Grossman, The case for cloud computing, IT Prof. 11 (2) (2009) 23–27.
[5] Seyedehmehrnaz Mireslami, Logan Rakai, Mea Wang, Behrouz Homayoun Far, Dynamic cloud resource allocation considering demand uncertainty, IEEE Trans. Cloud Comput. 9 (3) (2021) 981–994, http://dx.doi.org/10.1109/TCC.2019.2897304.
[6] Sivadon Chaisiri, Bu-Sung Lee, Dusit Niyato, Optimization of resource provisioning cost in cloud computing, IEEE Trans. Serv. Comput. 5 (2) (2011) 164–177.
[7] Google, Google cloud price calculator, 2014, https://cloud.google.com/products/calculator/.
[8] Transaction Processing Performance Council, TPC benchmark C (On-line trsansaction processing) specification, 2022, http://www.tpc.org/tpcc/.
[9] Erol Egrioglu, Cagdas Hakan Aladag, Ufuk Yolcu, Fuzzy time series forecasting with a novel hybrid approach combining fuzzy c-means and neural networks, Expert Syst. Appl. 40 (3) (2013) 854–857.
[10] Kehua Li, Zhenjun Ma, Duane Robinson, Jun Ma, Identification of typical building daily electricity usage profiles using gaussian mixture model-based clustering and hierarchical clustering, Appl. Energy 231 (2018) 331–342.
[11] Pedro Pereira Rodrigues, Joao Gama, Joao Pedroso, Hierarchical clustering of time-series data streams, IEEE Trans. Knowl. Data Eng. 20 (5) (2008) 615–627.
[12] Vit Niennattrakul, Chotirat Ann Ratanamahatana, On clustering multimedia time series data using k-means and dynamic time warping, in: 2007 International Conference on Multimedia and Ubiquitous Engineering, MUE'07, IEEE, 2007, pp. 733–738.
[13] Chonghui Guo, Hongfeng Jia, Na Zhang, Time series clustering based on ica for stock data analysis, in: 2008 4th International Conference on Wireless Communications, Networking and Mobile Computing, IEEE, 2008, pp. 1–4.
[14] Xishuang Dong, Lijun Qian, Lei Huang, Short-term load forecasting in smart grid: A combined cnn and k-means clustering approach, in: 2017 IEEE International Conference on Big Data and Smart Computing, BigComp, IEEE, 2017, pp. 119–125.
[15] Lalit Gupta, Dennis L. Molfese, Ravi Tammana, Panagiotis G. Simos, Nonlinear alignment and averaging for estimating the evoked potential, IEEE Trans. Biomed. Eng. 43 (4) (1996) 348–356.
[16] Vit Niennattrakul, Chotirat Ann Ratanamahatana, Shape averaging under time warping, in: 2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Volume 2, IEEE, 2009, pp. 626–629.
[17] Françis Petitjean, Alain Ketterlin, Pierre Gançarski arski, A global averaging method for dynamic time warping, with applications to clustering, Pattern Recognit. 44 (3) (2011) 678–693.
[18] Philip Virgil Astillo, Daniel Gerbi Duguma, Hoonyong Park, Jiyoon Kim, Bonam Kim, Ilsun You, Federated intelligence of anomaly detection agent in iotmd-enabled diabetes management control system, Future Gener. Comput. Syst. 128 (2022) 395–405.
[19] Irena Koprinska, Dengsong Wu, Zheng Wang, Convolutional neural networks for energy time series forecasting, in: 2018 International Joint Conference on Neural Networks, IJCNN, IEEE, 2018, pp. 1–8.
[20] Vinod Nair, Geoffrey E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Icml, 2010.
[21] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[22] Tania Lorido-Botran, Jose Miguel-Alonso, Jose A. Lozano, A review of auto-scaling techniques for elastic applications in cloud environments, J. Grid Comput. 12 (4) (2014) 559–592.
[23] Rui Han, Li Guo, Moustafa M. Ghanem, Yike Guo, Lightweight resource scaling for cloud applications, in: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, ccgrid 2012, IEEE, 2012, pp. 644–651.
[24] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, Patrick Valduriez, Streamcloud: An elastic and scalable data streaming system, IEEE Trans. Parallel Distrib. Syst. 23 (12) (2012) 2351–2365.
[25] Lucia Schuler, Somaya Jamil, Niklas Kuḧl, Ai-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments, in: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing, CCGrid, IEEE, 2021, pp. 804–811.
[26] Enda Barrett, Enda Howley, Jim Duggan, Applying reinforcement learning towards automating resource allocation and application scalability in the cloud, Concurr. Comput.: Pract. Exper. 25 (12) (2013) 1656–1674.
[27] Hamid Arabnejad, Claus Pahl, Pooyan Jamshidi, Giovani Estrada, A comparison of reinforcement learning techniques for fuzzy cloud autoscaling, in: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID, IEEE, 2017, pp. 64–73.
[28] Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, Automatedcontrol of multiple virtualized resources, in: Proceedings of the 4th ACM European Conference on Computer Systems, 2009, pp. 13–26.
[29] Ahmed Ali-Eldin, Johan Tordsson, Erik Elmroth, An adaptive hybrid elasticity controller for cloud infrastructures, in: 2012 IEEE Network Operations and Management Symposium, IEEE, 2012, pp. 204–212.
[30] Guofu Feng, Saurabh Garg, Rajkumar Buyya, Wenzhong Li, Revenue maximization using adaptive resource provisioning in cloud computing environments, in: 2012 ACM/IEEE 13th International Conference on Grid Computing, IEEE, 2012, pp. 192–200.
[31] Jing Jiang, Jie Lu, Guangquan Zhang, Guodong Long, Long optimal cloud resource auto-scaling for web applications, in: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, IEEE, 2013, pp. 58–65.
[32] Spyridon Chouliaras, Stelios Sotiriadis, Real-time anomaly detection of nosql systems based on resource usage monitoring, IEEE Trans. Ind. Inform. 16 (9) (2019) 6042–6049.
[33] Rodrigo N. Calheiros, Enayat Masoumi, Rajiv Ranjan, Rajkumar Buyya, Workload prediction using arima model and its impact on cloud applications' qos, IEEE Trans. Cloud Comput. 3 (4) (2014) 449–458.
[34] Sadeka Islam, Jacky Keung, Kevin Lee, Anna Liu, Empirical prediction models for adaptive resource provisioning in the cloud, Future Gener. Comput. Syst. 28 (1) (2012) 155–162.
[35] Ali Yadavar Nikravesh, Samuel A. Ajila, Chung-Horng Lung, Towards an autonomic auto-scaling prediction system for cloud resourceprovisioning, in: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, IEEE, 2015, pp. 35–45.
[36] Fei Xu, Fangming Liu, Hai Jin, Athanasios V. Vasilakos, Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions, Proc. IEEE 102 (1) (2013) 11–31.
[37] Yi Li, Fangming Liu, Qiong Chen, Yibing Sheng, Miao Zhao, Jian-ping Wang, Marvelscaler: A multi-view learning based auto-scaling system for mapreduce, IEEE Trans. Cloud Comput. (2019).
[38] Fei Xu, Fangming Liu, Hai Jin, Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud, IEEE Trans. Comput. 65 (8) (2015) 2470–2483.
[39] Xiaomeng Yi, Fangming Liu, Di Niu, Hai Jin, John C.S. Lui, Cocoa: Dynamic container-based group buying strategies for cloud computing, ACM Trans. Model. Perform. Eval. Comput. Syst. (TOMPECS) 2 (2) (2017) 1–31.
[40] Wei Deng, Fangming Liu, Hai Jin, Xiaofei Liao, Haikun Liu, Reliability-aware server consolidation for balancing energy-lifetime tradeoff in virtualized cloud datacenters, Int. J. Commun. Syst. 27 (4) (2014) 623–642.

Spyridon Chouliaras received his B.Sc. degree in Mathematics from University of Ioannina and his M.Sc. degree in Data Science from Birkbeck, University of London. He is currently working towards the Ph.D. degree in the Department of Computer Science and Information Systems, Birkbeck, University of London, United Kingdom. His research interests include cloud computing, auto-scaling, resource provisioning, anomaly detection and applied machine learning.

Dr Stelios Sotiriadis is a computer scientist working in the area of computing systems, working with algorithms that improve the performance of large scale systems. He enjoys exploring data science techniques that span the areas of distributed computing systems including Cloud computing, Internet of Things and applications of statistical learning algorithms for real-time big data analytics. His research directions are shaped around the design and implementation of new techniques and adoption of existing trends for the current systems to become more efficient, interoperable, and reliable and to operate on massive data sets. Dr Stelios Sotiriadis is a lecture of Computer Science at Birkbeck, University of London.