ADRL: A Hybrid Anomaly-Aware Deep Reinforcement Learning-Based Resource Scaling in Clouds

Sara Kardani-Moghaddam, *Member, IEEE*,
Rajkumar Buyya, *Fellow, IEEE*, and Kotagiri Ramamohanarao, *Member, IEEE*

Abstract—The virtualization concept and elasticity feature of cloud computing enable users to request resources on-demand and in the pay-as-you-go model. However, the high flexibility of the model makes the on-time resource scaling problem more complex. A variety of techniques such as threshold-based rules, time series analysis, or control theory are utilized to increase the efficiency of dynamic scaling of resources. However, the inherent dynamicity of cloud-hosted applications requires autonomic and adaptable systems that learn from the environment in real-time. Reinforcement Learning (RL) is a paradigm that requires some agents to monitor the surroundings and regularly perform an action based on the observed states. RL has a weakness to handle high dimensional state space problems. Deep-RL models are a recent breakthrough for modeling and learning in complex state space problems. In this article, we propose a Hybrid Anomaly-aware Deep Reinforcement Learning-based Resource Scaling (ADRL) for dynamic scaling of resources in the cloud. ADRL takes advantage of anomaly detection techniques to increase the stability of decision-makers by triggering actions in response to the identified anomalous states in the system. Two levels of global and local decision-makers are introduced to handle the required scaling actions. An extensive set of experiments for different types of anomaly problems shows that ADRL can significantly improve the quality of service with less number of actions and increased stability of the system.

Index Terms—Cloud computing, anomaly detection, deep reinforcement learning, performance management, vertical scaling

1 Introduction

Cloud model is a widely used computing paradigm for today's data and computing-intensive applications. The main concepts of virtualization and elasticity help the users to share computing resources on a pay-as-you-go model, taking into account the dynamicity of the workloads. Although the inherent characteristics of the cloud model offer the required flexibility for running dynamic applications, it can bring new challenges for the effective management of resources. The efficacy of resource management solutions can be interpreted from the level of user happiness; however, a combination of heterogeneity of applications, resource sharing conflicts, workload patterns, etc. can contribute to the violation of service level agreements (SLA) and users' Quality of Service (QoS). Therefore, proper scaling of resources depends on the comprehensive understanding of environmental changes and dynamic factors that can affect the performance of the system.

On the other hand, the workloads in the cloud are dynamic and uncertain. This is especially highlighted in applications that involve humans as their client-side users and therefore all the uncertainty from user's actions and their environment affects the application performance. Therefore, the prediction of the future workload is not easy and depends on many factors including human behaviors as users of the system (number of users, usage patterns, etc.), or application specifications and functionalities (new releases, bugs in the codes, etc.), some out of the knowledge of system administrators. Dynamic threshold-based solutions, time-series based analysis or machine learning-based techniques are proposed to address these problems [1], [2], [3], [4]. However, considering the uncertainty of the environment, it is critical to have a solution with a policy for updating the base assumptions, parameters, and learning models. Therefore, having an updatable decision-maker is an essential part to have an adaptable system with regard to the scaling of resources to ensure QoS satisfaction in the presence of various performance-related problems.

We have investigated adaptive learning frameworks such as reinforcement learning (RL) and how they can fit into our problem. In RL, continuous interaction of agents with surroundings develops an up-to-date knowledge base by collecting dynamic measurable metrics of the system. The knowledge is formulated as a set of states that define an abstract representation of the target system. RL is modeled as a control loop and gradual learning happens in a process of trial and error. The feedback-based gradual learning help to increase the autonomousness of the system and the ability to adapt to the changes. This feature is especially important in an uncertain environment, where the prior knowledge is not very clear. Therefore, at each step, the

• *The authors are with Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, VIC 3010, Australia. E-mail: skardani@student.unimelb.edu.au, {rbuyya, kotagiri}@unimelb.edu.au.*

Manuscript received 19 June 2019; revised 31 July 2020; accepted 15 Sept. 2020. Date of publication 22 Sept. 2020; date of current version 8 Oct. 2020.
(Corresponding author: Sara Kardani-Moghaddam.)
Recommended for acceptance by R. Tolosana.
Digital Object Identifier no. 10.1109/TPDS.2020.3025914

available knowledge is used to select actions that may change the environment. Then, the knowledge base is updated with the recent feedback from the environment.

While the RL paradigm seems to fit our problem, when the action should be triggered and the type of the selected actions are two main challenges that make the problem difficult in terms of the complexity and dimensionality of the state/action space. First of all, the level of resource control granularity considered in RL can target different types of performance problems. Despite many RL based attempts in the literature, the possibility of having a range of scaling actions including vertical and horizontal for different states of the system are not investigated. Second, the majority of RL based solutions do not consider the possibility of reaching a stable state where no action is required to move toward new states. In fact, the inherent characteristic of RL which learns from the results of triggered actions in the environment along with the highly dynamic nature of cloud and constraints on available resources can push the system to constantly change its state to observe the consequences of combinations of states and actions. While recent developments in Deep learning-based RL frameworks (DRL) try to utilize the learning capability of deep networks for modeling the value of state/action pairs, their focus is more on improving the efficiency of RL in searching larger state/action tables rather than the evaluation of the state value and the needs for taking new actions. Particularly, in the context of cloud computing resource management, the actions are meant to be triggered as a response to the performance problems in terms of resource utilization and QoS. These actions have consequences in terms of both cost and time and should be carefully selected considering the performance state of the system. This requirement highlights the need for more customized solutions that integrate the performance-related knowledge in the RL decision making process.

To address the above-mentioned challenges, we propose a deep reinforcement learning resource scaling framework that combines two levels of vertical and horizontal scaling to respond to the identified problems in the web-based applications under various performance anomaly issues that affect resource consumptions. The proposed solution utilizes an anomaly detection module to detect the persistent performance problems in the system as a trigger for the decision-making module of RL to perform a scaling action for correcting the problem. The deep learning part helps to increase the quality of decision making in large state space of the problem while the anomaly detection module addresses the timely trigger of scaling decisions. Two levels of scaling are proposed to address various types of performance problems including local VM-level resource shortage and system-level load problems. Accordingly, the *contributions* of this paper are as follows:

- Proposing an anomaly-based controller for RL decision-maker based on an isolation-based method to increase awareness of the performance state of the environment.
- Proposing a two-level scaling decision-maker as part of the action set in the RL framework.
- Proposing a Deep Q-learning based RL model to respond to the local anomalies of the system such as CPU and memory bottleneck problems. Accordingly, we propose state-based penalties of scaling decisions to speed-up the learning curve of RL.
- Performing extensive experimental evaluation of the proposed system under various loads and anomalous events. The experiments demonstrate ADRL ability to improve performance compared to the benchmark and state-of-the-art methods.

The rest of this paper is organized as follows: Section 2 overviews some of the related work in the literature. Section 3 discusses the motivation and assumptions in our modelings. Section 4 overviews the basics of Reinforcement Learning architecture. Section 5 presents a general discussion of the main components followed by the details of ADRL framework in Section 6. Section 7 presents the experiments and validation results. Finally, Section 8 concludes the paper with a summary and future works.

2 Related Work

While the dynamicity of the cloud environment with on-demand resources is bringing higher levels of flexibility for end-users, it makes the problem of resource management more complex. Resource scaling decisions are usually a response to the performance degradations of the system. However, a variety of factors at different levels of granularity from the workload and application-level characteristics to software and hardware functionality can affect the performance. Therefore, a proper solution should exploit adaptable models and decision-makers to create a self-directed learning environment.

The problem of autonomous scaling of resources can be easily mapped to MAPE-K architecture (Monitor, Analyze, Plan, and Execute over a shared, regularly updated Knowledgebase) of the autonomic systems. Following this architecture, [4] proposes a cost-aware auto-scaling framework with the focus on possible improvements at the execution level. The planning is done based on the threshold-based rules on monitored metrics to change the number of VMs in the system. While a threshold-based decision-maker is simple and convenient in terms of interpretation and implementation, the lack of the flexibility to adapt to the changes in the environment makes that a sub-optimal solution for these types of problems. To achieve higher adaptability at the decision-making level, Reinforcement Learning introduces a self-adaptable framework that can easily be matched by the phases of MAPE architecture. RL has been used for various types of resource management in the cloud. [5] utilizes Q-learning as part of the planning phase of the MAPE loop. The decisions are made as a combination of Markov decision table and Q-table to decide on adding/removing of VMs in the system. A fuzzified version of Q-learning and SARSA learning is introduced in [6]. They use the fuzzy rules on the monitored metrics as a solution to reduce the number of states and as a result the size of the Q-table. While these works use threshold and rule-based techniques to decrease the number of states, we take a modeling approach and create deep learning-based models to support a higher number of states. Moreover, the focus of our proposed is to investigate the efficacy of combining two levels of horizontal and vertical scaling for local anomalous behaviors.

TABLE 1
Related Works on RL-Based Cloud Performance Management

Work	Base RL	Resource Management Problem	Dimensionality Solution	Decision Level	Anomaly aware	Scaling Method
[5]	RL(Q-learn)	Scaling	Fuzzy states	Global	X	H
[6]	RL(SARSA, Q-learn)	Scaling	Fuzzy states	Global	X	H
[7]	RL(SARSA)	Scaling	Parallel agents, Function approximation	Global	X	H
[8]	RL(least-square policy iteration (LSPI))	Migration Management	Sparse Projection	Global	X	-
[9]	RL(Q-learn)	Migration management	-	Local (Host-level)	X	-
[10]	RL(Model-based approach)	Scaling	Decision-Tree based Models (Adaptive state partitioning)	Global	X	H
[11]	RL(SARSA)	Scaling	Model-based Environment	Local (Host-level)	X	V
[12]	RL(Q-learn)	Scaling	Parallel agents	Local (Host-level)	X	V
[13]	RL(Q-learn)	Task Scheduling	Deep RL , Multi-level Decision Maker	Global	X	-
[14]	RL(Q-learn)	VM to Server Mapping, Power Management	Deep RL, Representation Learning	Global, Local	X	-
Proposed work	RL(Q-learn)	Scaling	Deep RL, Multi-level Decision Maker	Global, Local	✓	H, V

Megh [8] is another RL-based system which targets the energy and performance efficiency of resource during live migrations of VMs in the system. The actions are defined as selecting the destination host of the migrated VMs. They use a projection method to reduce the state space complexity of their problem to a polynomial dimensional space with a sparse basis. Alternatively, Q-learning is used in [9] to schedule the live migrations of VMs. A combination of *waiting* and *migrating* actions are used to decide on the order of VM movements in the presence of network congestions to ensure having enough available bandwidth for on-time migrations. In contrast to these works, our work focuses on resource scaling actions that change the configuration of resources as a response to the performance problems in the system. [10] introduces an adaptive state-space partitioning technique to overcome the high dimensional state problem. The environment is represented as a global state at the beginning. Then, as more data is available, new states are created which maps the new observed behaviors of the system. This technique is especially important when the amount of training information is limited and the cost of collecting new data is high in terms of the time and operational costs. Alternatively, our work addresses this problem by having a distributed approach and utilizing Deep Reinforcement Learning to handle the local state of the VMs. Moreover, our work also focuses on the timing of decision making and the importance of using knowledge from local performance analytics to distinguish between global and local problems. VCONF [11] and VScaler [12] are two other frameworks that use the RL paradigm for vertical scaling of resources. VScaler uses a parallel learning technique where agents can share their experiences from the environment to speed-up the convergence. VCONF exploits neural networks (NN) learning to model the relation of (s, a) pairs with their corresponding rewards. The same parallelization technique as VScaler is also used by RLPAS for managing the number of VMs in the system [7]. The work presented in [15] targets temporal performance issues from co-located VMs by designing a parallel Q-learn model to decide on horizontal scaling actions. They consider both costs and QoS in modeling the reward function. The parallel agent assigned to each VM communicates with other agents to share the observations from the environment. Our work, however, targets the local and global performance problems and investigate the efficacy of proactive horizontal and vertical scaling actions in response to these problems.

The general idea of model-based RL as discussed in VCONF and the concept of Deep-RL enables the system to adaptively learn in complex problems with high dimensional space and low actions. Introduction of DRL techniques and their success in playing Atari offers new directions for the problem of dynamic, continuous-time state-space of resource management. Accordingly, DRL-cloud [13] is proposed to minimize the long-term energy cost of the data centers. The problem is formulated as a two-level task scheduling. The first level assigns tasks to a cluster of servers and the second phase chooses the exact VM on the selected server. Another work by [14] leverages DRL in a two-level VM allocation and power management framework. The DRL agent is used at the global layer for allocating VMs to hosts while RL and workload analysis are used in local VMs to manage the power. Our model is inspired by such models, but focuses on the hybrid scalings as part of the action set as well as anomaly-based triggering of the decision maker for decreasing the amount of oscillation resulted from sequential actions. Table 1 compares some of the RL based works in the literature considering their resource management actions and techniques for handling high dimensional state space.

3 Motivation and Assumptions

The elasticity feature of the cloud environment which allows to scale resources dynamically based on the performance of the system brings the flexibility to handle dynamic applications with constantly changing requirements. However, the dynamic adjustment of resources in accordance with the state of the system requires self-adaptable techniques that can interact with the environment and learn the effect of resource changes in a variety of load and resource configurations. To achieve this goal, the proposed solution should be able to answer three main questions.

The first question is when the decision should be made. Time-based monitoring and decision making [5], [12] is a common approach that helps the system to continuously adjust the amount of resources according to the load and performance state of the system. However, in the context of cloud resource management, the dynamicity of the environment can push the system to make unnecessary actions in response to the temporal performance problems. For example, a short spike in load can over-utilize the resources for a short amount of time. A proper response to over-utilization is to increase the amount of resources. While this is a correct action at the time of the observation, the problem is a temporal spike and the system quickly goes back to the normal load while the mount of resources is increased which may cause an under-utilized state. On the other hand, depending on the application, only some states require direct actions. For example, in the case of cloud-hosted applications, reaching a high resource utilization (close to defined thresholds) without SLA violations is recognized as a desirable state and requires no action. If there is no action, the system will continue working with the same amount of resources and the same pattern of workload until a change occurs. This change can be a result of workload variation, resource availability dynamics, or application behavior. When such change occurs, the learning continues by triggering new actions and observing changes of the states until another stable state is reached. Second question is which types of scaling should be actioned. Two main types of scaling in the context of VM resources are vertical and horizontal scaling. Vertical scaling changes the amount of allocated resources for one VM and horizontal scaling changes the number of VMs in the system. This decision is especially important for web-based applications which are shown to be prone to many local performance problems that impact CPU and memory resources [16]. While horizontal scaling can help for general load problems, it is shown that local problems can benefit more from vertical scalings in terms of the performance maintenance and resource utilization [17]. Finally, we should decide how to select an action for each state of the system. A common solution for this problem is a combination of if-then-else rules and threshold-based methods that describe the system in two main states of over-utilized and under-utilized and adjust the amount of resources accordingly. However, for a highly dynamic cloud environment, there are many internal and external factors such as CPU hog and memory leak problems that can affect the performance of the VMs. These types of problems require complex rules and analyzing methods to be properly managed. Considering the constant changes in application requirements as well as the limitation of physical resources which affect the amount of available resources, self-adaptable solutions show potential for automating the process of resource management. These systems can learn from the environment and tune their parameters, update their models, and adjust their decisions based on the most recent feedback from the system.

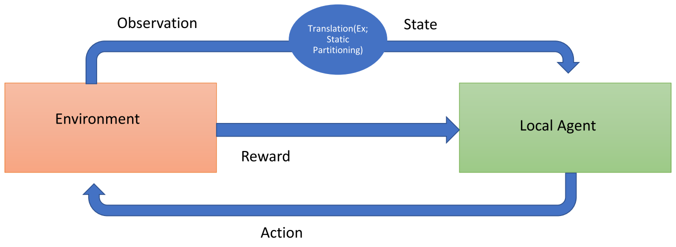


Fig. 1. Main components of general reinforcement learning framework.

Given the above explanation, we define our problem as the dynamic reconfiguration of resources in the cloud in response to the performance issues for applications under the local and global resource pressures which are discoverable from resource-level performance indicators and patterns. Particularly, we target web-based applications when the performance of their components are impacted by the client-side(human) behavior and cloud environment dynamics. These components can be individually scaled (stateless and loosely coupled components) as part of the cloud scaling solutions to respond to the increase in the workload. This work also addresses the experiences of end-users and considers QoS as a measure for validating the violations of user-level expectations. QoS metrics are application and user-dependent and are selected based on common indicators of user satisfaction. In this work, the solution is targeting service level providers who have access to the VM resources and VM-level performance metrics and define their SLA based on the user-perceived service response times.

4 Preliminary on Deep Reinforcement Learning Framework

Fig. 1 shows a general view of the RL framework for a problem which manifests the target environment. An agent is responsible to continuously monitor the environment and make observations of the important features. Collected observations are translated to one of the states s_i from the set $S = (s_1, s_2, \ldots, s_N)$. Each state represents an abstract description of the main features of the system. The goal of RL is to gradually learn how to move between states to maximize a long-term objective function in terms of the total rewards from each action. Each movement is done by selecting an action a_i from set $A = (a_1, a_2, \ldots, a_M)$. At each decision time t, the agent decides to perform action a_t based on the obtained knowledge from previous movements and the current state s_t. The environment sends back a scalar feedback (reward r_t) as the value of the action and its impact on the state of the system. These feedbacks are then used to update cumulative value of (s_t, a_t) pairs table. The gradual learning happens as a result of the many trial and rewards in the form $S * A -> R$ over time to achieve an optimal policy for the agent.

Q-learning is a type of RL for continuous time Semi-Markov Decision Problems (SMDP) with the goal to obtain an

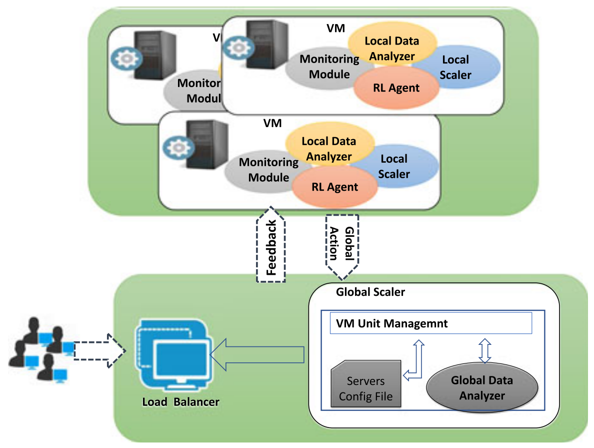


Fig. 2. General architecture of ADRL.

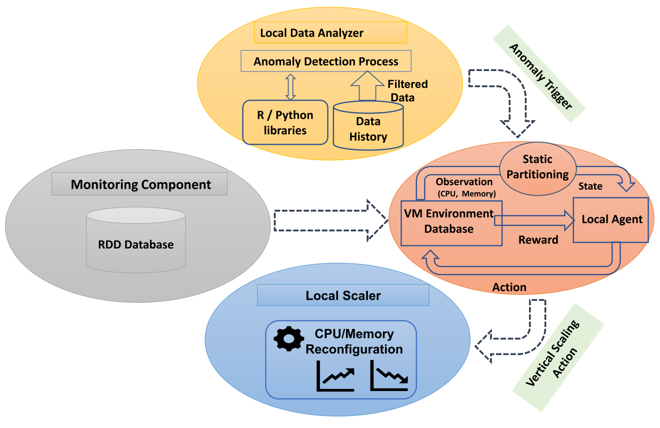


Fig. 3. The interaction among local ADRL components.

optimal policy Π to maximize value function $Q_{\Pi}(s,a)$ which estimates the accumulative discounted value of being in state s and performing action a under the policy Π. Suppose that at decision epoch t, action a_t is selected. At the next decision epoch $t+1$ and having the reward $r(s_t,a_t)$, the Q function value can be updated as follows:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha[r(s_t,a_t) + \delta max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)], \tag{1}$$

where $\alpha \in (0,1]$ is the learning rate and $\delta \in [0,1]$ is the discount factor. This formula defines a mapping table between state/action pairs to their expected values. However, defining and updating this table for large scale problems can be challenging as the size of the table exponentially increases with increasing states and action spaces. Deep reinforcement learning combines the goal-directed optimization of RL approach with Deep Neural Net (DNN) based approximation of expected values. DNN based function approximators are used to learn to predict the value of those actions with regard to our environment. Networks can be trained and modeled offline to iteratively find proper weights and minimize the loss functions. Trained models are then used during online RL execution to select best actions based on the state of the environment.

5 System Design

Fig. 2 depicts a high-level view of the main components of ADRL and their interactions with the external user and cloud environment. The users send their requests to the load balancer component which distributes them among existing active VMs. Fig. 3 shows the details of 4 main modules in each VM as described in the following:

- *Monitoring Module* which is responsible for monitoring the measurable features of the environment. In the context of VM monitoring, these features can be resource utilization measurements such as CPU and memory.
- *Data Analyzer (DA)* performs data cleaning and behavior modeling of the VM. The aim is to create and continuously update an abstract model of VM performance and detect unexpected violations. The detected anomalies identify the occurrence of performance problems and the need for corrective actions.
- *DRL Agent* is the main decision-maker that is triggered after identifying an existing anomaly in the system by the data analyzer module. It takes the observations from the monitoring module of the system as input. The output of this module is an action that defines some changes in the configurations of resources. The selected action is fed to the local scaler or sent back to the global layer for further processing.
- *Local Scaler* is responsible for performing actions that define some type of change in resource configurations of corresponding VM.

Algorithm 1 shows the main steps of the ADRL framework. Each VM monitors the performance of its resources by collecting resource utilization metrics at regular time intervals. The collected data are fed into both local DA and RL agent for processing. DA utilizes feed-forward Neural Networks to perform a prediction of the future values of each collected metric. Then, the predicted values are used as input of an anomaly detection algorithm to decide if the system is behaving abnormal compared to the performance models from previous observation of VM. If an anomaly event is detected, the DRL component is triggered to decide on a corrective action based on the observed state of the system. In this work, the performance anomaly detection is defined in favor of end-users and points to the events that can violate the expected Quality of Service objectives. As a result, the anomaly event is defined as continuous and unusual changes in the values of VM performance metrics such as CPU and memory utilization which can affect the ability of the machine to process user requests in an acceptable time. Finally, it should be noted that the DRL agent can also be triggered as a result of exceeding the maximum *Time Between Actions (TBA)*. This condition is included for cases when the performance is in a normal state, but the resources are under-utilized. Although no anomaly is triggered during normal times, we want to give the decision-maker a chance to move toward states with higher utilization (possibly by removing extra resources).

Upon receiving the selected action from DRL, it is checked that the action is a local resource scaling request or not. If the answer is yes, the local scaler is called to adjust the amount of allocated resources based on the requested changes. On the other hand, if the action is a global scaling request, the results are sent back to the global scaler which is responsible for controlling the number of VMs. The global scaler can decide on adding new VMs to reduce the total amount of resource utilization in the system or shutting down the existing VMs to reduce under-utilized VMs and resource wastage. While the action is executing, the system enters a *Locked state* when no new action is performed. This strategy gives the system enough time to adapt to new configurations and reach a stable state. The details of each step and corresponding algorithms are explained with more details in the following section.

Algorithm 1. ADRL: General Procedure

```
1: Initialize $Q(s,a)$ table with profiled transitions from testing
   experiments;
2: Initialize anomaly detection Models;
3: while The system is running and at the beginning of perfor-
   mance-check interval do
     /* This part of the code is executed locally in
     each VM */
4:   $u_t^p \leftarrow$ Predict the performance indicators for $vm_i$ at time $t$
     based on the monitored data at time $t-1$
5:   $s_t \leftarrow$ Identify the performance state for $vm_i$ at time $t$
     based on the predicted values $u_t^p$
6:   if $s_t$ shows an anomaly then
7:     Increase the $counter$ by 1;
8:   end
9:   if ($counter \geq L$ AND $vm_i$ is not in Locked state) OR
     $Time(a_{t-1}) \geq TBA_{max}$ then
10:    Call DRL Agent for a new Action $a_t$;
11:    Execute $a_t$ following Algorithm 2;
12:    Schedule an update for learning model to be done
       according to Algorithm 3;
13:  end
14: end
```

Algorithm 2. ADRL: Execution Phase

```
  input : $A_t$: Selected action at time $t$
1: while The system is running and at the beginning of perfor-
   mance-check interval do
     /* This part of the code is executed locally in
     each VM */
2:   if $A_t$ is local then
3:     Initialize all indicators in $f$ to 0;
4:     for $a_j \in A_t$ do
5:       if $a_j$ is a request of change for resource $j$ then
6:         $R_j^{new} = R_j^{old} + a_j * R_j^{unit}$
7:         if $R_j^{min} \leq R_j^{new} \leq R_j^{max}$ then
8:           Apply the change
9:         end
10:      end
11:    end
12:  end
13:  else
       /* This part of the code is executed in the
       master node */
14:    $Add$ new VMs or $Remove$ from existing VMs based on
       acceptable utilization and state of the environment.
15:  end
16: end
```

TABLE 2
Description for Notations

Notation	Description
R_j	Amount of resource j
R_j^{unit}	Unit of change for resource j. For example, one core for CPU resources
TBA_{max}	Maximum allowed time between actions
$V(s_t)$	Value of the state s_t
u_j	Utilization of resource j.
a_t	Action at time t
rt	Response Time
L	Minimum number of violations before the system reacts to an anomalous event

Algorithm 3. ADRL: DRL Agent

```
  /* Select an Action */
1: $s_t \leftarrow$ Performance state at time $t$ based on monitored data
2: Choose an action from set A randomly with $\epsilon$ probability,
   otherwise select an action with maximum Q value;
  /* Perform scheduled learning */
3: if Learning schedule is triggered then
4:   $s_{t+1} \leftarrow$ Performance state at time $t+1$;
5:   Calculate $r_t$ based on Equation (5);
6:   Store stransition ($s_t$, $a_t$ ,$r_t$, $s_{t+1}$ ) in VM profile memory $M$;
7:   Update Q according to Equation (1);
8: end
```

6 ADRL: A Deep RL Based Framework for Dynamic Scaling of Cloud Resources

In this section, we detail the main components of the ADRL framework. As explained in Section 3 and Algorithm 1, ADRL is composed of three main parts to address the identified challenges in an adaptable resource management solution. We should note that this is a general architecture and each part can be easily extended to new data analysis techniques, more advanced resource management solutions such as migrations of VMs, and other mapping techniques to select among state/action pairs. Table 2 presents a list of notations used in this paper.

6.1 Deep Reinforcement Learning (DRL) Agent

The DRL module addresses the mappings of states to actions where a proper scaling action should be selected for the current state of the system. Let us assume we have a pool of active VMs $V = (v_1, v_2, \ldots, v_P)$ as our global environment. Each vm_i is described with a tuple $U = (u_{i1}, u_{i2}, \ldots, u_{iK})$ where u_{ij} is a scalar value representing the utilization of resource type j on vm_i. For each resource type j, an action a_j can be performed. If a_j is greater than zero, it corresponds to increasing resource j by amount a_j; If it is zero, it means the resource is unchanged and negative values correspond to amount of released resources. Therefore, depending on the total number of types of resources, the final set of the actions for each VM is defined as the Cartesian product of the sub-action sets of its resources as follows:

$$A = \Pi_{j=1}^{K} A_j$$

where A_j is the set of all possible actions for resource j.

Accordingly, the purpose of the DRL agent is to find a proper configuration of resources by continuing changes of respective resources and receiving feedback on the outcomes of the changes. However, the changes of resources on vm_i are limited to the minimum amount of allocated resources for a VM as well as the available resource of the host machine. Suppose a scenario where the environment $V = (v_1, v_2)$ is handling the daily load of a web application with normal utilization of resources. The dynamics of workload during the day is handled by adding/removing resources for each VM asynchronously. Then, during a peak period, the load drastically increases which causes unexpected over-utilization of resources. In this scenario, the system is facing a situation that adding resources at the local level may not be enough. Therefore, we add a special action a_{global} to the action set A where a_{global} corresponds to a request for help from global layer. Section 6.3 discusses these actions in more details.

DRL Agent $->$ Action Selection: Upon receiving an anomaly alert, DRL agent is called to choose an action in response to the detected performance problem. Let us assume that s_t is the observed state of the performance anomaly. In order to choose an action from the action set, we need a policy that exploits the available knowledge from the feedback of previous decisions (exploitation) and also tests new actions to improve the knowledge of state/action relations (exploration). We use a dynamic version of ϵ-greedy policy which is a standard policy for having a trade-off between exploration and exploitation policies. ϵ-greedy policy selects a random action with a probability equal to ϵ, otherwise it selects an action with maximum Q value in the table. In order to have a dynamic policy with a higher exploration at the start, ϵ is initialized with 1 and as the number of observed states increases the value of ϵ decreases until it reaches a minimum value [13].

DRL Agent $->$ Learning-Model Update: When the system applies an action, a waiting time is required so the effect of changes can be reflected in the environment. At this time, the DRL agent calls for an update based on the newly observed state s_{t+1}. The agent first stores the transition (s_t, a_t, r_t, s_{t+1}) in a profile memory. Then, the reward is calculated for the pair (s_t, a_t) to evaluate the goodness of the selection.

The final purpose of ADRL is to improve the QoS and utilization of services. Therefore, the reward is formulated according to this goal and is composed of three components as follows:

- *QoS*: The Quality of Service describes the level of satisfaction from the user perspective. Considering web application characteristics, performance problems that cause unexpected changes at the resource-level can eventually impact the response time of the whole system. Therefore, the goal of the system is to track the resource-level performance patterns and trigger actions that avoid the degradations of user-level experience. Depending on the application and user agreements, there are a variety of QoS metrics such as availability, reliability, throughput, makespan, etc. We choose response time (RT) as a measure for this metric. RT represents the waiting time for each request from the submission to the completion including runtime and queuing times. Therefore, the processing delays caused by resource shortages are dynamic parts of delay values which are included in the metric and should be minimized. Let rt be the average response time of requests during time interval t to $t+1$. Then, the reward of rt (R_{rt}) is calculated based on Equation (2) where RT^{max} and RT^{min} are maximum and minimum acceptable values. The minimum value condition is added to consider cases that an application goes into an unresponsive or overloaded state and no request can be accepted and as a result, RT drops to a near-zero value. When rt is a value between the min and max thresholds and therefore satisfying SLA, the reward will always be 1. When rt violates the thresholds, the utility function will decay to zero. In other words, R_{rt} utility function punishes any action that causes SLA violations.

$$R_{rt}(rt) = \begin{cases} e^{-(\frac{rt-RT^{max}}{RT^{max}})^2} & rt > RT^{max}, \\ e^{-(\frac{RT^{min}-rt}{RT^{min}})^2} & rt < RT^{min} \, . \\ 1 & otherwise, \end{cases} \quad (2)$$

- *Resource Utilization*: While having an under-utilized environment can give the users a high QoS in terms of the running time of requests, the wastage of resources is not acceptable for service owners. Wasted resources increase costs in terms of the monetary value as well as energy wastage in the environment. Therefore, we need to consider the resource utilization for each resource j of vm_i in the final reward value. This value helps the decision-maker to move toward decisions that increase the utilization of resources while considering the satisfaction of user expectations through QoS value introduced in the previous part. Equation (3) defines this value as an average of utilization on all resources where ${U_j}^{max}$ defines the maximum acceptable utilization for corresponding resource j and $u_j \in (0, 1]$. Higher utilization makes positive impacts on the final reward value. However, if the utilization is violating the maximum threshold (the second part of the Equation (3)) the value of R_{ut} starts increasing which negatively impacts the final reward.

$$R_{ut}(u_j) = \begin{cases} \frac{\sum_{j=1}^{N} {U_j}^{max} - u_j}{N} + 1 & u_j \le {U_j}^{max}, \\ \frac{\sum_{j=1}^{N} u_j - {U_j}^{max}}{N} + 1 & otherwise \end{cases} . \quad (3)$$

- *State Transitions Value*: While running the experiments with ADRL, we noticed that a sequence of (s, a) transitions can lead the decision-maker to be trapped in a loop between states. This can happen as a result of the simultaneous changes of resources by the actions that are affecting the value of more than one resource. This is especially important for applications where changes of one resource have a dominant effect in terms of the utilization compared to the others. Suppose we have vm_i with two resources CPU and memory in an under-utilized state. Action $a = \{-a, +a\}$ is triggered and one unit of CPU is removed while one unit of memory is added. Since the application is a CPU sensitive one, the utilization of CPU significantly increases while memory shows a small change. Although the utilization of memory is still in the under-utilized state, this action can result in good reward value. Therefore, differentiating among transitions with utilization improvements of one resource can be challenging. Although this observation can be dependent on the units of changes and the characteristics of applications, considering the dynamicity and heterogeneity of cloud-hosted applications this behavior can be expected. As a solution for this problem, ADRL introduces a state value function and transition penalty as Equation (4) where function V assigns manual weights to the states. The aim is to include some domain knowledge on top of the environmental feedbacks from the system. The definition is in accordance with the concept of reward shaping for boosting the learning phase with application-dependent information [18], [19]. The undiscounted potential-style shaping helps to evaluate the value of transition from one state to another based on the $P : S \times A \times S- > R$ form. If an action is causing a transition from a higher value state to lower ones, a penalty value is considered in the final reward function. In contrast, moving from a lower state to higher states affects the reward value positively. Accordingly, the penalty function is defined as follow:

$$P(s_t, s_{t+1}) = \begin{cases} 1 & if V(s_t) < V(s_{t+1}), \\ -1 & if V(s_t) > V(s_{t+1}),. \\ 0 & otherwise \end{cases} \quad (4)$$

Finally, Equation (5) shows the final value of $r(s_t, a_t)$ pair as the total rewards in terms of the QoS, utilization and state value changes. Higher values of R_{rt} and lower values of R_{ut} increase the final reward.

$$r(s_t, a_t) = \frac{R_{rt}(rt)}{R_{ut}(util)} + P(s_t, s_{t+1}). \quad (5)$$

Having all the information from transition (s_t, a_t, r_t, s_{t+1}) ready, updating of the Q-table can be done based on the new information and Equation (1). In order to improve the stability of learning and parameter updating in the presence of anomaly and temporal spikes which introduce abnormal transitions, we leverage experience replay as a sampling technique during training. This technique uses the profile of the past transitions to randomly select mini-batches of records to be used for training the learning networks. Random selection of records also helps to overcome the correlation among sequential experiences as well as improving the efficiency by using each experience in many of the updates [20].

6.2 Anomaly-Aware Decision Making

In the context of cloud resource management, the actions are triggered as a response to the performance problems in the system. However, the base DRL loop usually works as a periodic decision-maker with iterative selection and updating steps to gradually adapt to the environment. Proactive event-based decision making is another approach where the decisions are made as a response to possible predicted performance problems. This helps the system to reduce the frequency of decision makings which also reduces the possibility of oscillation among states. In order to achieve this goal, we choose a fast and memory-efficient anomaly detection algorithm called IForest [21]. IForest is based on isolation-tree (iTree) data structures where each tree is constructed by randomly choosing attributes and dividing the instances based on a random value for the corresponding attribute. To better understand the iTree structure, suppose we have the performance observations of vm_i for last t logging intervals as $X = (U_1, U_2, \ldots, U_t)$. The first step is to randomly select a sample of ψ instances from X where $\psi < < t$. This sample is used to build the first iTree. To create the root node, a random attribute from U is selected and the sample instances are divided into two subgroups based on their value for the selected attribute. The new subgroups create two sub-nodes of the root node. This process continues for each sub-node until a termination criterion is reached. It is shown that the iTree structure isolates anomaly instances in shorter branches of the tree and therefore the path length of the tree from root to the leaf nodes represents a comparable value for evaluating the degree of anomalousness for each instance [22].

IForest model is built based on an ensemble of many iTrees and the anomaly scores are average of path length on all trees. Having a worst time and space complexity O($T\psi^2$) and O($T\psi$) for the training of T iTrees, IForest is a promising option for dynamic environments where the models require regular updates to capture the latest state of the system. Moreover, iTrees can be built independently which makes it easy to have parallel implementations of algorithms to have more efficient anomaly detection. ADRL leverages IForest as the core of anomaly detection module where the performance observations are used to train and initialize models and future performance values are tested to identify the states that are violating the recently observed behavior of the system.

One point worth mentioning here is that the triggering of an anomaly state can be a result of a change between states in terms of the values of monitored metrics from workloads and VMs. Three problems arise as a result of this transition to be addressed.

First, the transitions among states can be a result of temporal spikes which can be expected in highly dynamic environments. To address this problem, one anomaly alert is not taken as a serious anomaly event. In fact, the DRL agent is triggered for making a decision when a continuous anomaly event is identified by receiving at least L consecutive alerts (Algorithm 1, Lines 4-7). Therefore, the system ignores the first few alerts to avoid unnecessary reactions to transient changes. The value of L can be decided based on a combination of factors such as system logging interval, application characteristics, and the degree of fault tolerance.

Second, if the transitions are real, the trained anomaly detection models may not reflect the new states and therefore there will be many false anomaly alerts. To solve this problem, we use an idea introduced in [17] for deciding the proper time for updating of the models. The idea is that the system can be classified into three separate states of transition, changed, and normal. In the transition state, the system observes many anomaly alerts and newly collected data are different in pattern/value compared to the training data. When the transition completes and system observes higher stability in monitored data, the system has moved to a new changed state. In our case, an update will happen in the changed state and therefore the new observations are representing new behavior of the system.

The prediction interval should be selected to consider the delay of actions to be effective in the system. In this case, we select an interval equal or larger than the maximum time required to finalize a scaling action which includes starting a new VM in the system and updating load balancers to add the new server in the list of active, schedulable resources. Finally, it should be noted that while the frequency of decision making is reduced by replacing the periodical triggering with anomaly triggers, we should still consider that not all decision epochs require a change in the states. If the performance is in a good state in terms of the reward values, no-change actions may give a better chance of reaching an optimal condition. Action a_j equal to zero as discussed in section 6.1 helps the system to experience the no-change effect on the performance of VMs.

6.3 Two-Level Scaling

As we explained in Section 6.1, two levels of scaling are considered in this work. The first level is defined for each resource of VMs. Three types of action as defined by a_j are applied based on the units of change for each resource. Let us assume one CPU core as the unit of the change for this resource. Therefore, $+a$ action increases the number of cores by a while $-a$ action removes a cores from the VM. Similarly, the unit of memory changes can be set as $256\ MB$ and therefore each action changes the amount of allocated memory with multiples of this unit. In our work, one unit is selected for each change. Moreover, the action is valid if the requested changes are not violating the available resource of the host machine or minimum acceptable amount of the allocatable resource to each VM.

The second level of scaling is performing horizontal changes at the global scaler which is responsible for managing the units of VMs and can change the number of VMs according to the state of the system. The global scaler has access to the utilization of all VMs. ADRL designs the global layer as a threshold based horizontal scaling algorithm. In an under-utilized environment, where the total resource consumption of VMs is lower than the threshold, the global scaler identifies VMs which have low utilization and starts deactivation process. Similarly, when the scaler finds an over-utilized state, where the total resource consumption is higher than the threshold, new VM is added to help reduce the load on existing machines.

Combining these actions, the system should be able to learn the relation between performance states with appropriate resource management decisions. Let us assume the anomaly detector module is predicting impending memory pressures (memory consumption higher than maximum thresholds) for one VM. Suppose system selects CPU scaling action as a response. Adding one new core, if not violating the available resources, is not helping the memory shortages for the application and RT will be impacted gradually. Therefore, feedback from the environment causes negative rewards for the combination of observed state and action. Similarly, selecting a global level action will trigger the global scaler which monitors the overall performance of the environment. However, as the total resource consumption is not violating thresholds, this state is considered as a local problem and no action is performed (no horizontal scaling). We expect to see gradual QoS degradations while moving between non-memory relevant actions. Finally, when the system selects a memory scaling action, the system adds $256MB$ to the VM memory which helps to respond to the resource pressures on the VM. Moreover, every action is followed by a locked-down period (locked state) which allows the system to reach a stable state and the effects of the selected action are reflected in the feedback results.

7 Performance Evaluation

In this section, the performance of the proposed framework is evaluated using CloudSim discrete event simulator [23]. An extension of CloudSim is used that includes analytical performance models of a web application benchmark [24] and an anomaly injection module [17]. The simulator helps us to create a controlled environment for performance anomaly testing and corresponding validations for different types of problems. We have tried to make realistic assumptions by considering the analytical characteristics of system behavior and including them in the modeling. These include the VM booting-time delays, session analytical models, temporal spikes, etc. Through these experiments, we demonstrate how the integration of proactive alerts of anomaly detector, knowledge from feedback-based RL system, and also scaling actions can respond to the local and global anomalous events.

7.1 Experimental Settings

We model the environment as a data center with two types of application and database server VMs. The configuration of VM templates for application server is one virtual core, 256 MB of Ram and Linux operating system and the maximum limit for resources are 5 cores and 3,072 MB, respectively. The workloads are based on the web-based user requests on Rice University Bidding System (RUBiS) benchmarking environment which models an auction site following ebay.com model. The application models interactive users performing browsing, searching, and performing transactions. As a result, it demonstrates the high variable demands of the cloud. It is also shown that this type of application is prone to many performance anomalies that involve CPU and memory resources. We use this characteristic to create local performance problems resulting from resource bottlenecks [16]. The modeled workload consists of both browsing and updating actions which result in a modification of the database. Each session of the web workloads is modeled based on the monitored resource usages of real requests on RUBIS [24]. To generate the performance models of the system, four attributes CPU, memory, disk utilization, and number of sessions are collected. VM startup times are also modeled based on the study done in [25]. The anomaly detection module is initialized by generating iTrees models for each VM. Unless otherwise specified, the value of parameters in IForest configurations and model updating schedules are according to the recommended settings as explained in [17]. A proper value for L can be selected considering the trade-off between computation overheads, the stability of the environment, and the performance degradation tolerance. Small values of L may cause the system to perform unnecessary checks of the performance or decide on preventive actions for many false alarms, while large values of L increases the time it takes for the system to start a scaling action in response to the performance problems. Based on our observation of scaling action delays to be effective (VM booting times) and system logging interval in the system, we have selected value $L = 6$ for this variable. Based on experimental observations, this gives the system enough time to avoid many temporal spikes originated from dynamics of the workload. In order to initialize the Q-table of the DRL agent, we run CloudSim for 48 hours and record the transitions and corresponding rewards in a file. These records are then used in a batch learning process to initialize the Q values [20]. We consider utilization of two resources, CPU and memory, to create state space. Unexpected patterns in each variable can represent a local anomaly involving one metric and the combination of those two can show load increase performance problems. The action space is combinations of three levels of vertical scaling as discussed in Section 6.1 for both CPU and memory and horizontal scaling action. For Deep Q-learning we use a constant learning rate $\alpha = 0.05$ value and a discount factor $\gamma = 0.9$. The number of layers is 20 and the size of mini-batches for profile memory is 50 based on our experimental evaluations. ϵ is initialized equal to one and decreased from 1 to 0.1 to give higher exploration capability in the initial iterations of learning with ϵ-greedy policy. The threshold values for RT are 0.1 and 2 and are defined based on the simulated application and workload characteristics.

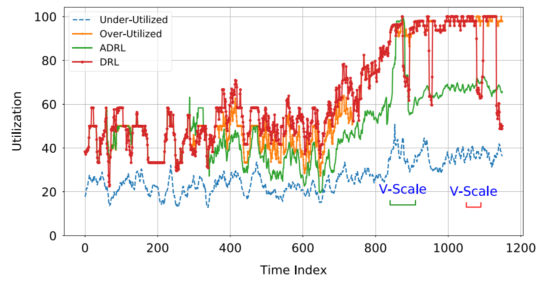
In order to assign weights to states for penalizing process, we follow a simple idea based on the static partitioning of the state space. Therefore, for each resource, the utilization is divided into 5 partitions and the incoming state values are mapped to the corresponding partition. Partitions with higher utilization get higher weights. DRL agent is implemented in the Python environment with TensorFlow and a wrapper is created to connect Java-based CloudSim simulator to python codes.

Each experiment has a duration of about 24 to 48 hours. These values are chosen to sufficiently capture application behavior for various scenarios. The normal workload is based on the RUBiS benchmark and the sessions are generated based on Poisson distribution with a time-based frequency as explained in [24]. Two types of CPU and memory anomalies are generated in CloudSim to create an increasing trend effect in the consumption of CPU and memory without significant changes in the normal load of the system. These anomalies start after the model initializations and at random times during execution. To create the increasing load effect, after 10 hours of normal load, the number of sessions start to increase in two phases by adding 5 and 20 sessions at each time unit, respectively.

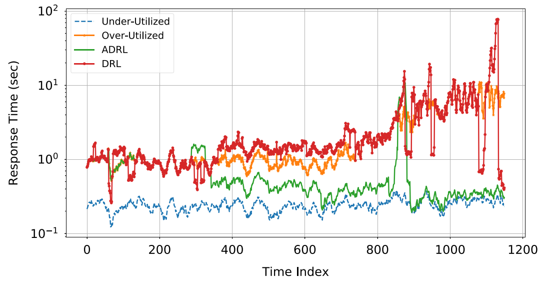
7.2 Experiments and Results

In order to evaluate the performance of ADRL, two static methods and one DRL based method are considered. In *Under-Utilized* method, the VMs are configured so that the total amount of allocated resources is more than the demanded ones. Therefore, with an under-utilized method, the user can experience the best QoS. In *Over-Utilized* case, the VMs are set up based on the minimum VM template configurations as described in Section 7.1 such that during the run of the experiment and by starting anomaly events the utilization of resources exceeds the acceptable level and some violations are allowed. In both cases, no scaling is done through the experiments, therefore generating a sample of the best and worst results to evaluate the general functionality of ADRL. We also implement a non-anomaly aware RL based algorithm similar to the approaches such as [11]. To have a fair comparison, we compare with an enhanced algorithm by implementing a Deep Learning based RL decision-maker with both vertical and horizontal scaling actions and name it as DRL to study the effect of anomaly-based decision making of ADRL.

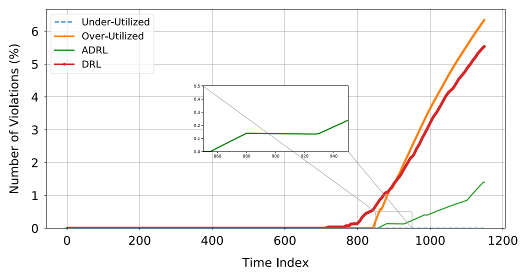
Fig. 4 presents the results of all methods on a workload with the CPU hog problem. The first diagram shows the CPU utilization corresponding to each scenario. As we can see, the under-utilized environment shows the lowest CPU utilization while over-utilized one has the highest utilization. While CPU consumption is increasing, both DRL and ADRL try different types of actions. These actions are not always the optimal choices that are expectable as the system is observing new states that may have a few historical records of their transitions before. However, as the system starts to violate the QoS around $t = 800$, both algorithms try to optimize the resource utilization by adding new cores to the VM. At this point, ADRL observes a transition in the utilization values, updates the anomaly detection model, and enters a stable state. The stability of the process can be seen around observation $t = 900$ and onward where no anomaly is triggered and therefore no action is performed to change the states. In contrast, DRL continues time-based decision making which may return the system back to the violation state. Although choosing $a_j = 0$ action can help the system to keep the current state, but some actions which are resulted from random selections or due to the temporal spikes of the performance can cause wrong changes of configurations and extra violations. These violations are also shown in the last graph of Fig. 4. This diagram shows the cumulative percentage of violations during each time interval. As the picture shows, ADRL can reduce the incremental results of QoS violations in the presence of anomalous behavior by performing vertical scalings and keeping the system in the normal state. In contrast, DRL can not show stable results in terms of violation reductions as it continuously returns the system back to an abnormal state. As already mentioned, this behavior is due to not recognizing



(a) CPU Utilization

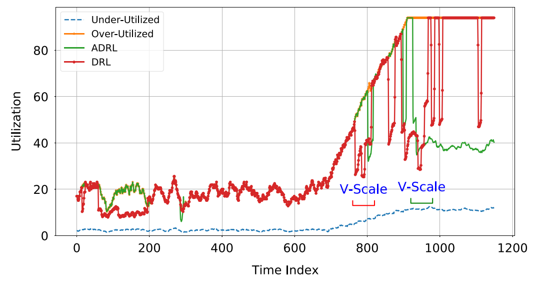


(b) Response Time

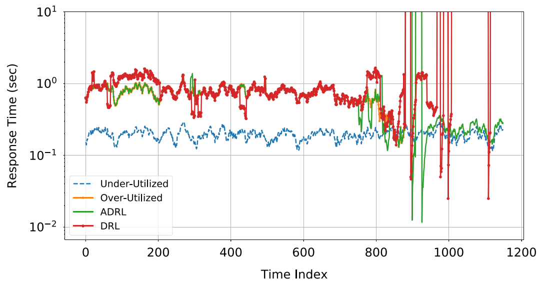


(c) SLA violations for CPU anomaly

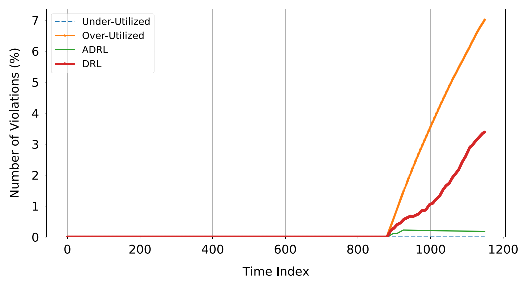
Fig. 4. CPU Utilization, Response Time (Log), and violations number for CPU shortage dataset. Time index indicates the sequence of logging points in the system. ADRL is able to pro-actively trigger vertical scaling actions in response to anomaly events (utilization more than 80 percent). It also shows higher stability in comparison to DRL with multiple changes of state between anomalous and normal states.



(a) Memory Utilization



(b) Response Time. Vertical upward spikes show the violations of SLA in terms of response time. ADRL can effectively decrease the number of violations by deciding to add more resources and keep the system in stable state while time-based decision making by DRL is returning back the system to an anomalous state by constant moves among states.



(c) Total percentage of violations. As the graph shows, with the start of anomaly and violations of SLA, ADRL adds extra resources which avoids further increases in the failed sessions.

Fig. 5. Memory Utilization, Response Time, and cumulative violations in the presence of memory shortage dataset. Time index indicates the sequence of logging points in the system. ADRL is able to pro-actively trigger vertical scaling actions in the response to anomaly alerts which decreases RT violations and rejected sessions.

the continuity of the anomaly state and trying to make new changes to maximize rewards with regard to the resource utilization.

Fig. 5 shows the utilization and RT diagrams for memory shortage problems in the system. To generate the anomaly state, after $t = 600$, a steady increase of the memory utilization is started and the results of each scenario for memory utilization and RT are presented. The diagrams for the under-utilized scenario does not show any significant change as there is still plenty of free memory available. In contrast, over-utilized execution gets affected immediately as the utilization exceeds corresponding thresholds which are reflected in the second diagram where RT shows sudden increases. These unexpected increases which are shown as vertical upward lines in the graph happen when the VM does not have enough memory and therefore becomes unresponsive while rejecting many of the new incoming requests. However, with the start of memory anomaly and an increase in RT violations, ADRL decided to add extra resources which avoid further violations as well as decrease the number of failed sessions. DRL, in contrast, achieves an initial decrease of RT violations by adding more resources; however, the time based triggering of decisions and sudden spikes of utilization while moving between states cause wrong actions that release some of the resources. The sequence of these add/removal of resources causes several violation spikes and returning the system back to the anomaly state. This is again due to the ignoring of the stability of the system in terms of being in an identified continuous anomalous state and particularly is expected when the system is experiencing higher explorations. For example, this can happen when the system is observing rarely seen states such as memory utilization higher than 30 percent in a CPU-intensive application. ADRL, however, correctly identifies anomaly states and after two wrong configurations,

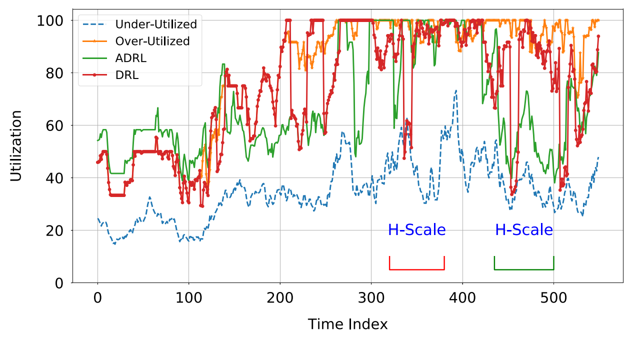


Fig. 6. CPU utilization for an overloaded system. Multiple scaling actions are performed during the running of DRL and ADRL algorithms. Two horizontal scaling actions done by ADRL and DRL methods are shown as an example.

around $800 \leq t \leq 900$ brings the system back to a steady performance. The last diagram of Fig. 5 demonstrates the results of the cumulative number of violations which highlights the ability of ADRL to reduce the total number of violations after detecting the anomalous behavior with regard to the memory utilization.

In order to show the response of the system to high load problems and triggering of horizontal scaling actions, we run CloudSim with a workload that increases the load to saturate resources. Fig. 6 shows the corresponding CPU utilization of this load and the changes made in the system for static and dynamic scenarios.

As we expect, the under-utilized run shows the lowest utilization, while the over-utilized configuration soon reaches the saturation point of resources. Both DRL and ADRL trigger a mix of vertical and horizontal scalings during their run. The horizontal scaling decisions that add new VMs for DRL and ADRL are shown with red and green marks on the diagram, respectively. However, the sequence of decisions made by DRL during the transitions of the system from abnormal to normal state weakens the expected effects of added VM in the system. The reason is due to the decisions that remove some cores from existing VMs which can temporally reflect increases of the utilization. However, the increase is happening during the transition of the system when the load is still increasing which as a result causes the violations of performance. In contrast, ADRL correctly identifies the continuous anomaly events and the number of decisions in the presence of temporal spikes is less and more accurate.

Fig. 7 shows the number of decisions corresponding to the scaling actions for both methods ADRL and DRL. As we have mentioned before, DRL includes a periodic decision-maker while ADRL triggers scaling actions as the response of detected anomalies. As a result, ADRL can significantly decrease the number of scaling actions. This reduction is important in the cloud environment as every scaling is changing the patterns of the performance in the system and therefore affecting the accuracy and updating interval of prediction models.

Finally, to validate the effect of penalty values of the reward function (Equation (5)) in guiding the decision-maker to higher value states, we run two versions of ADRL with penalties included (ADRL_WP) and without that (ADRL_NP). The results of this experiment are shown in

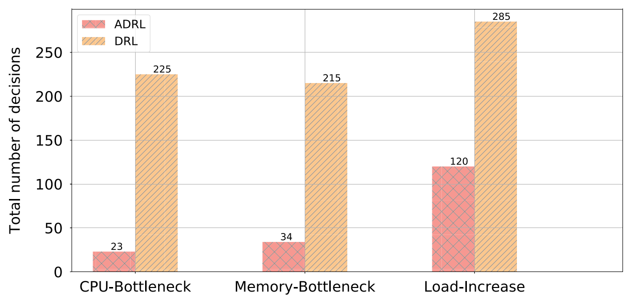


Fig. 7. Total number of decisions (scaling actions) for both methods DRL and ADRL for each dataset. ADRL can decrease the number of decisions with an event-based decision-making process. Time index indicates the sequence of logging points in the system.

Fig. 8. As we can see, ADRL_NP selects more action types that increase resource allocations and moves the system to the states with lower utilization which as described in Section 6.1 have lower value in accordance with the reward function. For example, there are a series of decisions to add resources around $t = 300$ or between $t = 600$ to $t = 900$ which reduces the utilization. However, by each reduction, the utilization part of the reward function reflects the negative effect of these movements which helps the system to recover (as it is shown around $t = 1000$) after a few steps. However, ADRL punishes the decisions that move the system to low utilization states while encouraging toward decisions that remove resources when the utilizations have not reached their maximum thresholds. Therefore, the general behavior of the system under ADRL management is more toward high utilization states with higher values as long as the SLAs are respected. This helps the system to quickly learn about the actions which configure resources to achieve higher reward values.

8 Conclusion and Future Work

In this work, the problem of VM-level resource scaling in cloud computing is addressed which includes dynamic changes of resource configurations to regulate the performance of the system. A variety of techniques such as threshold-based rules and time-series analysis are proposed in the literature as a solution to the problem. These techniques try to define models that determine when and what type of scaling action should be performed. However, considering the constantly changing environment of cloud, self-learning paradigms are required to be able to interact with the environment. In this work, ADRL is proposed as

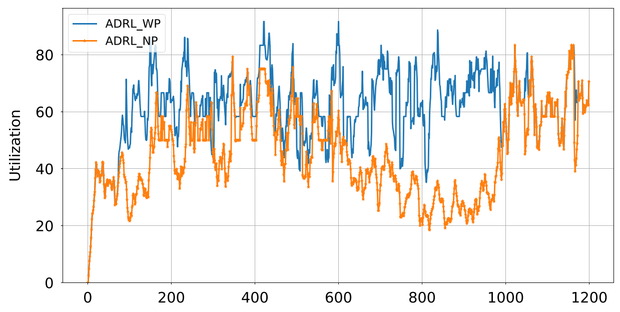


Fig. 8. A comparison of CPU utilization with two versions of ADRL. ADRL_WP performs a penalizing process as part of the reward calculation while ADRL_NP ignores this step.

a two-level adaptable resource scaling framework. ADRL models the problem of resource scaling as a Deep Reinforcement Learning framework with the capability of observing the performance of surroundings and taking actions as a response to the problems. ADRL identifies performance problems by using an anomaly detection model and the actions are a combination of horizontal and vertical scaling changes. We show that the ADRL framework can achieve better results in terms of identifying and correcting performance problems with a smaller number of decisions. Moreover, it is shown that different types of performance anomalies can be addressed by scaling decisions at various levels of granularity.

As part of the future work, we plan to extend the current framework to consider the energy consumption and cost of the resources in their decisions. This can be added as a new layer of decisions where the amount of the change (positive or negative) is decided by a low-level decision-maker while a high-level agent decides on the best choice of the scaling in terms of the cost and energy considering the global state of the system. Moreover, the current experiments are working on a web-based workload benchmark with an auction-based profile. Other types of applications and more recent web-based technologies might create different varieties of resource consumption profiles as well as resource anomaly signatures. These new profiles and their impacts on the complexity of the ADRL decision-maker (anomaly detector module and resource-level actions) needs to be investigated.

References

[1] J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen, "Workload predicting-based automatic scaling in service clouds," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, 2013, pp. 810–815.

[2] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "PREPARE: Predictive performance anomaly prevention for virtualized cloud systems," in *Proc. 32nd IEEE Int. Conf. Distrib. Comput. Syst.*, 2012, pp. 285–294.

[3] X. Li, A. Ventresque, J. Iglesias, and J. Murphy, "Scalable correlation-aware virtual machine consolidation using two-phase clustering," in *Proc. Int. Conf. High Perform. Comput. Simul.*, 2015, pp. 237–245.

[4] M. S. Aslanpour, M. Ghobaei-Arani, and A. N. Toosi, "Auto-scaling web applications in clouds: A cost-aware approach," *J. Netw. Comput. Appl.*, vol. 95, pp. 26–41, 2017.

[5] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," *Future Gener. Comput. Syst.*, vol. 78, pp. 191–210, 2018.

[6] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *Proc. 17th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2017, pp. 64–73.

[7] J. V. B. Benifa and D. Dejey, "RLPAS: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment," *Mobile Netw. Appl.*, vol. 24, pp. 1348–1363, 2019.

[8] D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan, "Learn-as-you-go with Megh: Efficient live migration of virtual machines," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2608–2609.

[9] M. Duggan, J. Duggan, E. Howley, and E. Barrett, "A reinforcement learning approach for the scheduling of live migration from under utilised hosts," *Memetic Comput.*, vol. 9, pp. 283–293, 2017.

[10] K. Lolos, I. Konstantinou, V. Kantere, and N. Koziris, "Adaptive state space partitioning of Markov decision processes for elastic resource management," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 191–194.

[11] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "VCONF: A reinforcement learning approach to virtual machines auto-configuration," in *Proc. 6th Int. Conf. Auton. Comput.*, 2009, pp. 137–146.

[12] L. Yazdanov and C. Fetzer, "VScaler: Autonomic virtual machine scaling," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, 2013, pp. 212–219.

[13] M. Cheng, J. Li, and S. Nazarian, "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *Proc. 23rd Asia South Pacific Des. Autom. Conf.*, 2018, pp. 129–134.

[14] N. Liu *et al.*, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 372–382.

[15] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency Comput., Pract. Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[16] B. Subraya, *Integrated Approach to Web Performance Testing: A Practitioner's Guide*. Pennsylvania, USA: IGI Global, 2006.

[17] S. K. Moghaddam, R. Buyya, and K. Ramamohanarao, "ACAS: An anomaly-based cause aware auto-scaling framework for clouds," *J. Parallel Distrib. Comput.*, vol. 126, pp. 107–120, 2019.

[18] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proc. 16th Int. Conf. Mach. Learn.*, 1999, pp. 278–287.

[19] S. Devlin and D. Kudenko, "Theoretical considerations of potential-based reward shaping for multi-agent systems," in *Proc. 10th Int. Conf. Auton. Agents Multiagent Syst.*, 2011, pp. 225–232.

[20] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[21] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 8th IEEE Int. Conf. Data Mining*, 2008, pp. 413–422.

[22] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 1, pp. 3:1–3:39, 2012.

[23] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[24] N. Grozev and R. Buyya, "Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments," *The Comput. J.*, vol. 58, no. 1, pp. 1–22, 2013.

[25] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 423–430.