Journal of Grid Computing (2022) 20: 15
https://doi.org/10.1007/s10723-022-09603-4

# Scalable Virtual Machine Migration using Reinforcement Learning

**Abdul Rahman Hummaida · Norman W. Paton · Rizos Sakellariou**

Received: 12 July 2021 / Accepted: 2 March 2022 / Published online: 28 April 2022

**Abstract** Heuristic approaches require fixed knowledge of how resource allocation should be carried out, and this can be limiting when managing variable cloud workloads. Solutions based on Reinforcement Learning (RL) have been presented to manage cloud infrastructure, however, these tend to be centralized and suffer in their ability to maintain Quality of Service (QoS) for data centres with thousands of nodes. To address this, we propose a reinforcement learning management policy, which can run decentralized, and achieve fast convergence towards efficient resource allocation, resulting in lower SLA violations compared to centralized architectures. To address some of the common challenges in applying RL to cloud resource management, such as slow learning and state/action management, we use parallel learning and reduction of the state/action space. We apply a decision making approach to optimize the migration of a VM and choose a target node to host the VM in such a way that brings response time within SLA level. We have also demonstrate unique, multi-level reinforcement learning cooperation, that further reduces SLA violations. We use simulation to evaluate and demonstrate our proposal in practice, and compare the results obtained with an established heuristic, demonstrating significant improvement to SLA violations and higher scalability.

**Keywords** Reinforcement learning · Data centre scalability · Virtual machine migration · Hierarchical architecture · Distributed architecture

A. R. Hummaida (✉) · N. W. Paton · R. Sakellariou
Department of Computer Science, University of Manchester, Kilburn Building, Oxford Rd, Manchester M13 9PL, UK

A. R. Hummaida
e-mail: abdul.hummaida@postgrad.manchester.ac.uk

N. W. Paton
e-mail: norman.paton@manchester.ac.uk

R. Sakellariou
e-mail: rizos@manchester.ac.uk

## 1 Introduction

Cloud computing is an established paradigm to give end users access to computing resources through a simplified as-a-service model. Cloud Providers (CPs) build data centres and abstract resources through a virtualisation layer, with a Virtual Machine (VM) as a common form. End users request these services through APIs, that map requests to virtual resources that reside on physical resources in the data centre.

VM placement, both to schedule initial virtual machines and to adapt the placement, to meet assorted goals, has been a subject of extensive investigation [17, 33, 37, 45, 50]. To perform VM placement, specific resource utilisation or compatibility requirements are typically required, such that a VM is mapped onto a physical node within the data centre. From within the pool of physical nodes that satisfy VM constraints, the mapping process becomes an optimisation problem that aims to increase resource utilisation, energy consumption or profit. Post initial VM placement, it is common for the cloud environment to undergo a load change, where the initial VM constraints are no longer met. CPs use adaptation methods to continuously monitor and perform VM placement [23].

VM placement is accepted as an NP-Hard problem and heuristic solutions have been used to solve it [61]. However, data centres are becoming increasingly large, which means the problem of making globally appropriate placement decisions is increasingly challenging. Many of the solutions to manage cloud infrastructure are centralized and suffer in their ability to support data centres with thousands of nodes. Furthermore, heuristic approaches have been shown to have scalability challenges [26, 53], in their ability to execute the decision making process with the increasing size of data centre infrastructure. Different approaches have been proposed to address this problem of scale [11, 44, 48, 56, 62]. These tend to be decentralized heuristics, with no central controller, and have been shown to manage a large number of nodes. A key challenge with heuristics is that their performance depends on multiple factors including the statistical patterns of resource demands, and if the underlying scenario changes, heuristics may start to perform poorly [21].

In this paper, we build on a hybrid architecture that has been shown to have benefits of rapid decision making [24], fewer SLA violations, lower network traffic utilisation and improved scalability as the number of nodes in the data centre increases, compared to centralized, hierarchical and other decentralized architectures. The hybrid architecture has hierarchical decentralized controllers operating at different scopes. On the lowest level, controllers dynamically adjust resource configurations and cooperate with other nodes and higher level controllers in performing VM placement. However, although the hybrid architecture supports localised decision-making in a global context, this still raises the question as to the policies, which map VMs to data centre resources, that should be applied at different levels in the hierarchy. The most suitable policy may depend on subtle features of the infrastructure and the workload, hence there may be benefits from learning the policy. Reinforcement Learning (RL) is an approach that develops or refines a policy in the light of experience [54]. In RL, an agent performs learning by interacting with an environment, and learning through trial and error. The agent takes actions and observes the outcome of these actions. RL has been applied successfully in a variety of resource management settings [15, 51, 55]. Here we develop an RL approach for the hybrid architecture [24] and apply it to cloud resource management. Multiple RL agents operate in a decentralized way and share the learning from migration's actions. Additionally, the agents uniquely cooperate at multiple management hierarchy levels to achieve rapid decision making and learn an optimal online policy. Our approach is shown to reduce SLA violations and achieve high scalability.

The contributions of this paper are as follows:

1. A realisation of a RL strategy in a specific hybrid architecture, with lower SLA violations and high scalability compared to a Heuristic based approach.
2. The utilisation of the hybrid architecture, to achieve unique, parallelised multi-level RL agent cooperation.
3. An empirical evaluation of the RL strategy in comparison with a heuristic strategy that has been shown to be effective in practice [7]. This shows the proposed RL approach improves SLA violation performance, compared to the heuristic approach, and further improvements are achieved when RL is combined with the hybrid architecture from our earlier work [24].

The rest of this paper is organised as follows. We first describe some of the challenges in designing an efficient resource management controller. Section 3 describes related work in cloud resource management, and Section 4 describes a background into RL. Section 5 summarises the hybrid architecture from our earlier work, and Section 6 describes our proposed RL approach. Section 7 presents an evaluation of our implementation and compares it to a heuristic hybrid, RL centralized and a heuristic centralized approaches. In Section 8 we draw conclusions and discuss future work.

## 2 Problem Statement

Cloud Providers (CPs) provide access to resources that are typically pooled and shared with multiple customers, with a layer of orchestration that separates individual customer usage. Physical resources are abstracted through virtualisation technology into compute, memory, storage and networking with logical separation of these resources, and typically presented as a Virtual Machine (VM). CPs customer workloads experience variability and VMs are created and deployed onto physical nodes to run customer workloads. CPs need to re-optimize the infrastructure regularly to provide high levels of availability and reliability.

An illustrative use case is VMs running web applications, such as e-commerce systems, which are typically N-tier and with web servers that process business logic. The resources assigned to the VM and the amount of incoming requests determine the *response time* experienced by end users. The web server will typically experience a variable arrival rate of requests. Using Fig. 1 as an example, Web Server 1 can experience an increase in requests, which will increase the resources used by Server 1 and increase the load on the physical node, and in turn could start to impact the response time for Web Server 2 and Web Server 3. Continuing with this state could have an adverse affect on all of the web servers. CPs aim to react to such conditions in order to preserve the customer's SLAs. A CP's goal is to enable such VMs to operate at a rate that meets customer agreed SLAs, and balance this with the CP's operating costs. We research and solve the following problems:

- Detect when a VM is stressed.
- Identify which VMs to migrate.

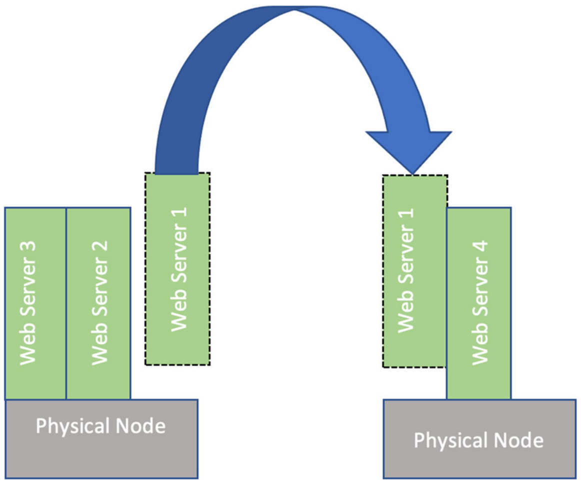


**Fig. 1** Managing SLAs

- Apply a decision making approach to optimize the migration of a VM, and choose a target node to host the VM in such a way that brings response time within SLA levels.
- Develop an architecture for the control system that monitors and optimises the migration of VMs.

In particular, we aim to solve these problems in ways that scale to work on data centres with thousands of nodes. Our goal is to design a scheme that minimises the SLA violations, by detecting stressed VMs and actioning a migration plan to choose new target nodes to house the VMs. One option is to move stressed VMs to a newly switched on node or a lightly loaded node, however this could have a negative impact on the amount of energy consumption in the data centre and thus impact CP's operating costs. Therefore, the optimisation scheme needs to balance meeting SLAs with energy consumption.

Data centre workloads vary, with new workloads being provisioned and other workloads being deprovisioned, so an efficient controller needs to cope with these varying demands. Additionally, web servers typically exhibit varying traffic loads that can cause nodes to become stressed and lead to SLA violations. When this occurs, the controller needs to engage a migration plan to reduce SLA violations.

In a large data centre managing thousands of VMs and nodes, a control scheme needs to be scalable and continue to be responsive. Decentralized architectures that have autonomous monitoring, management and feedback are well suited to large scale environments [58]. In this paper, we have developed a control approach where each node operates autonomously and contains a learning agent that shares learnt knowledge with other agents.

## 3 Related work

This section outlines related work on VM placement and on the use of machine learning, and in particular reinforcement learning (RL) for resource management.

VM Placement is mapping of customer workloads to infrastructure resources, in a way that achieves a particular objective, such as reducing energy consumption or load balancing, while ensuring SLAs are met [34]. VM placement consists of two parts: *initial* VM placement, which refers to the first allocation of VMs to nodes in the data centre, and VM *migration* or relocation, which involves the revision of an earlier placement decision. VM placement performs the mapping in order to meet an SLA, energy or profit goal and has been studied extensively.

Meeting an SLA objective can be seen as detection of an overloaded node, selection of VM(s) on the overloaded node, identification of a target node, and engaging the hypervisor to migrate the VM to the identified target node in order to avoid performance degradation. When a node is deemed overloaded and the adaptation process chooses to migrate a VM from a stressed node, VM(s) need to be chosen for migration. VM selection approaches include: Minimum Migration Time chooses a VM that takes the shortest time to complete the migration, Random Selection chooses a VM based on a uniformly distributed discrete random variable and Maximum Correlation selects a VM of the highest correlation of CPU utilization with other VMs to migrate [7]. In this paper, we use a similar approach to Maximum Correlation to select a VM for migration, by sorting the VMs and choosing the VM with highest correlation.

Heuristic approaches are widely used in the literature for VM migration. The authors in [22] proposed an approach to increase efficiency of node utilization and balance utilization of CPU and memory usage on active nodes across the data centre. The approach used a multi-dimensional resource usage model for target node selection and used it to guide the VM placement process. A resource usage factor is assigned to each node and used in node selection. Their experiments show minimisation of low utilized resources and more balanced utilization of CPU and memory usage on active nodes. Uniquely, the authors in [20] considered joint VM and container migration. The approach divides the cloud resource management problem into sub-problems including over-load/under-load detection, identifying if a VM or a container should be migrated, VM/container selection and migration of the VM/container. Local Regression was used to detect overloaded nodes and VM selection was done using Minimum Migration Time [7]. Target nodes for migrations were selected using SLA-aware allocation. The authors in [63] proposed a multi-constraint optimization model by considering migration cost and remaining runtime of VM migration, and used a heuristic policy. The applied constraints were the total CPU/memory requirements of VMs allocated should not exceed the node's resource capacity and a VM should be assigned to a single physical node; maximum duration that a node can be in SLA violation and the remaining runtime for a VM were also considered. However, heuristics typically do not find a globally optimal answer but may provide locally optimal outcomes [42, 67].

Cloud environments are highly complex, and are typically multi-tenanted with non linear workloads; as a result they experience high variability. Machine Learning (ML) techniques can offer an opportunity to adjust resource management in a dynamic way, which is reflective of the context of cloud environments [32]. ML techniques can be categorised as *Supervised Learning* where every data sample is labeled and used as input. The learning process works by associating features of the input and human feedback. In *Unsupervised Learning* samples are used as input, but unlike supervised learning, there are no labels and the learning process aims to learn the data distribution within the sample. For example VM usage patterns can be used to cluster VMs into distinct groups through unsupervised learning. In *Reinforcement Learning* there is no labeled input, instead an agent learns dynamically from its environment and balances exploration of new knowledge versus exploitation of known knowledge.

Some ML approaches focus on auto-scaling resources, autonomously provisioning and deprovisioning resources. The authors in [41] presented an auto-scaling method for adaptive provisioning of elastic cloud services, based on ML time-series forecasting and queuing theory, aimed at optimizing response time. The approach uses Support Vector Machines (SVM) to predict the average node load for the following hour, and then use this with a queuing model to adjust the resources assigned to a node. Their experiments show SVM has better prediction than moving average and linear regression. Similarly, another prediction approach was presented in [64] with Long Short Term Memory (LSTM) time-series prediction, and provisioning through queue theory. Their results show LSTM performed better in terms of prediction accuracy than the SVM and Autoregressive Integrated Moving Average. A Neural Network technique was presented in [60], which proposes an adaptive selection that can choose a VM consolidation approach based on the current environment and the cloud provider's priority on energy and SLA violation. The approach firstly generates a raw dataset by simulating the methods for several time steps. Each row will contain the initial environment parameters and normalized evaluation result of all policies. The results (energy and SLA violations) for each row then are normalized. A performance score is calculated using the evaluation priority and normalized evaluation result from the raw dataset; this score is then used to train the neural network. Another framework for resource reservation is presented in [52], based on load prediction and several ML approaches including Neural Networks, Linear Regression, RepTree and M5P. The approach takes an initial reservation plan and monitoring data as inputs and optimises the plan based on monitoring data from observations, with CPU being the main monitored resource. The evaluation showed RepTree was able to learn faster than the Neural Network; however, the Neural Network ultimately yielded better predictions.

While ML models approaches have shown to be effective, one limitation of these approaches is that they are typically trained offline and require retraining to make use of new data [49]. Cloud environments are dynamic and exhibit regular changes in the structure of workloads and access patterns. Aptly, RL can operate online and learn dynamically from interacting with a changing environment and make use of new information to enhance the decision making process. Additionally, RL approaches do not require prior knowledge of the optimization model and are not coded explicit instructions relating to which action to take next; instead, they learn actions through feedback from the environment. These features make RL well-suited to cloud resource management resource management [41].

Auto-scaling of the assigned VMs is the focus of some of the approaches in the literature, by using RL to add more resources for customer workloads. The authors in [28] propose a general purpose model-free learning algorithm, based on Q-learning, that adapts to unknown system specifics, such as application traffic, to generate scaling up or down actions. Our proposal also uses Q-learning, however we focus on migrating stressed VMs as opposed to auto-scaling. To speed up the convergence of RL, the authors in [6] developed an approach that parallelises Q-learning to speed up convergence of agents in order to achieve auto-scaling of VMs. We propose a similar approach of parallel learning and further enhance it with cooperative learning between agents running at different layers of a hierarchical cloud infrastructure. The problem of autonomous scaling of cloud resources can be mapped to MAPE-K architecture (Monitor, Analyse, Plan, and Execute) [19]. The approach enhanced the performance of the planning phase and a planning module uses linear regression to predict future demands, with Q-learning performing dynamic resource allocation. Their experiments show the approach increases the resource utilization and decreases the total cost while reducing SLA violations. Our proposal has many similarities with monitoring, planning and execution modules, although, unique to our approach, is cooperative learning between RL agents running at different layers of a hierarchical infrastructure.

Reinforcement learning techniques can suffer from the curse of dimensionality, where the state and/or action space grows exponentially, which introduces challenges in the time needed for the RL agent to explore a given environment and introduce space complexity in memory consumption by the agent. To address this, some approaches utilise function approximation, such as Deep Q-Learning (DQN) [40], which is an approach of combining deep learning and Q-learning to combat the challenges of Q-learning in environments with a large or continuous state action space. The authors in [30] propose a DQN based model to respond to anomalies in CPU and memory bottlenecks and apply granular actions to autoscale resources. The approach in [9] also proposes a Deep Q-learning based approach to adjust the size of a cluster, by taking the state of the cluster as input and training an RL agent to resize a cluster based on administrator defined policies and rewards. The agent can use Deep Q-learning, Double Deep Q-learning or Full Deep Q-learning, and the approach was compared to other RL and decision-tree based approaches and shows it gains rewards up to 1.6 times better. Alternatively to using a DQN, the approach in [46] used a coarse-grained Q-table and can achieve higher resolution in the Q-table with less cost. The approach proposed granular actions to adjust CPU and memory resources, and applied it in a distributed learning mechanism using Q-learning. The work in [10] used a heuristic method to reduce the state space to a smaller set, by dividing the original state space into multiple exclusive subsets, where a range of states can fit into the same subset, thus reducing the state space to aid RL convergence speed. Other non-statistical approaches for function approximation have been proposed [5, 8, 25]. To address the curse of dimensionality, instead of function approximation we used an aggregation approach in our proposal. This reduces states and actions into smaller groups, with multiple states being mapped into a smaller number of states and actions.

Migration of stressed VMs, which are failing a quality of service metric, to another target node, is the focus of some of the work in the literature and has a similar aim to our proposal. The authors in [47] propose a Deep RL based framework that performs VM migrations and uses a proximal policy optimization (PPO) algorithm and a neural network based on LSTM for function approximation. The architecture of the approach is split onto an offline and an online part. The offline part trains an agent by sampling log data, which is generated by the online agent. The online agent has a similar method to the offline, except it does not update the agent parameters. Online decision information is used for the next offline training. Our approach also caters for reducing SLA violations, however we choose a different state action reduction approach to manage the challenges with Q-learning. Additionally, the approach was only evaluated with a small number of VMs, while our experiments were evaluated with thousands of VMs and nodes. The authors in [43] propose a Q-learning controller to manage complex workload arrival patterns and use a decentralized architecture, with each node responsible for maintaining its own SLA performance. The approach is able to add nodes and scale down by shutting down excess nodes to save on energy consumption. To combat the state space challenge in Q-learning, the approach uses a reduced state space. Similarly, our proposal uses a decentralized architecture, applies knowledge sharing among the RL agents, uses aggregation to reduce the state action space, and uses linear regression to monitor QoS metrics like response time. However, the uniqueness of our approach is cooperative learning between RL agents running at different layers of a hierarchical cloud infrastructure. The authors in [66] investigate VM migration during data centre upgrade and use a DQN to decide a target node for each VM migration with the objective of minimising the total migration time. We use a state action aggregation approach to address the dimensionality challenge, while the authors use function approximation. Table 1 presents a summary of the approaches used in VM cloud resource management, including heuristics and ML techniques.

While there have been attempts at examining the scalability of approaches based on RL [10, 46], these tend to be at a small scale that is not representative of the size of the infrastructure in public clouds. We propose a highly scalable RL approach and examine its ability to manage a large infrastructure, with many thousands of nodes.

## 4 RL Background

Reinforcement Learning (RL) is a machine learning technique that enables an agent to learn within an interactive environment, through trial and error, and uses signals from the environment in a feedback loop. In Fig. 2, the agent monitors the current state of the environment (Step 1), and chooses an action from the available options on the environment (Step 2). The environment will then generate a reward for the action taken by the agent, and transition to a new state (Step 3). The goal oriented agent aims to learn the set of actions, a policy, that will lead it to a specific goal, or to maximise an objective function. RL problems are typically formulated with well defined transition probabilities and modelled as a Markov decision process (MDP) [54].

While RL has shown much promise, there are significant challenges applying to practical real world problems [16], including limited offline training logs, learning on the real system where exploration has to be limited and delays in the system actuators to gather action reward.

RL approaches are categorised as model based or model free methods, depending on whether full knowledge of the model can be specified. Model based approaches need knowledge of the environment model, while model free methods learn a policy based on observations and rewards [54].

There are two common control categories of RL. *Value-based or off-policy methods*: RL algorithms proceed to learn the value function for every state/state-action pair to arrive at the optimal policy. Q-learning is a common algorithm in this category. *Policy-based or on-line methods* directly learn the parameters for the policy, instead of learning an

**Table 1** Adaptation proposals

| MA | Objective | Considered resource | | | | Techniques |
|---|---|---|---|---|---|---|
| | | *CPU* | *Memory* | *Storage* | *Network* | |
| Yadav [65] | SLA & Energy | x | | | | Heuristic |
| Gholipour [20] | SLA & Energy | x | | | | Heuristic |
| Xu [63] | SLA & Energy | x | x | | | Heuristic |
| Gupta [22] | Energy | x | x | | | Heuristic |
| Vozmediano [41] | SLA | x | | | | SVM & QT |
| Moghaddam [30] | SLA | x | x | | | DQN |
| Bitsakos [9] | SLA | x | x | x | | DQN |
| Ren [47] | SLA & Energy | x | x | | | DQN |
| Ren [66] | SLA | x | x | x | | DQN |
| Witanto [60] | SLA & Energy | x | | | | Multiple |
| Sniezynski [52] | SLA & Energy | x | | | | Multiple ML |
| Bibal [8] | SLA | x | x | | | SARSA |
| Arabnejad [5] | SLA | x | | | | Q & SARSA |
| Bu [10] | SLA | x | x | | | Q |
| Ghobaei-Arani [19] | SLA | x | x | x | x | Q |
| Rao [46] | SLA | x | x | | x | Q |
| Nouri [43] | SLA | x | x | | | Q |
| Jamshidi [25] | SLA & Energy | x | x | | | Q |
| Barrett [6] | SLA | x | x | x | | Q |
| Hummaida (This paper) | SLA | x | x | | | Q |

Key: Q: Q-learning, DQN: Deep Q-Learning, ML: Machine Learning, SVM: Support Vector Machines, QT: Queueing Theory, SARSA: State–Action–Reward–State–Action

explicit policy function, by fine tuning a vector of parameters in order to select the best action to take for policy. SARSA is a common example in this category.

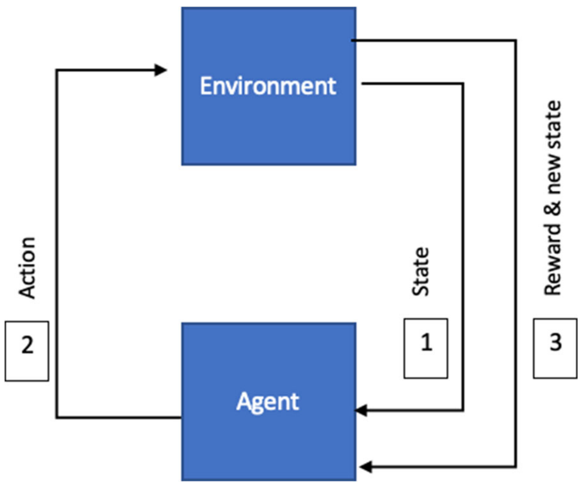


**Fig. 2** RL continuous process

Deep reinforcement learning combines RL with Deep Neural Network based approximation of expected values. An offline phase prepares the network with prior system knowledge, for example from execution. These are then used during online RL execution to select best actions based on the state of the environment [35].

Q-learning [59] is a common control strategy in cloud resource management, due to its simple implementation. Q-learning is model free RL algorithm, belonging to a collection of algorithms known as Temporal Difference (TD) methods. Q-learning estimates the optimal action value function, independent of the policy being followed, and does not require a full model of the environment. The action-value function or Q-function is updated using (1), and approximates the value of selecting a certain action at a certain state.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Max\, Q((s_{t+1}, a_{t_{+1}}) - Q(s_t,\ a_t))] \tag{1}$$

In this equation, $\alpha \in[0,1]$ is the learning rate, or step size, and determines how the agent learns from recent updates. A high value for $\alpha$ means the most recent information obtained is utilised while a low value implies slower learning. To dampen the reward's effect on the agent's choice of action, the discount factor $\gamma \in [0,1]$ is used. When $\gamma$ is set to 1, the agent will emphasise greater weight to rewards in the future. When it is closer to 0, the agent will only consider the most recent rewards. $Max\,Q(s_{t+1}, a_{t+1})$ returns the maximum estimate for the future state-action pair. Once the Q-Value is calculated it is then stored in the agent's Q-Table.

One of the challenges in action selection is exploration vs exploitation, which is the challenge of choosing to further explore the environment for possibly better rewards or choosing to exploit the known reward paths. To speed up the process of learning an effective policy, the agent needs to exploit rewarding actions, but it needs to also find these online. A common approach is $\boldsymbol{\epsilon}$-greedy [54], which selects the action with the highest estimated reward most of the time. With a small probability of $\boldsymbol{\epsilon}$, we choose to explore, and not exploit by randomly selecting an available action, independent of the action-value estimates we have previously learned. Other action selection methods include soft-max and optimistic initialisation of values [54].

Some of the challenges with RL include poor initial performance, large training time, and large state space. To improve the poor initial performance, human experts can set initial values for a given state/action [38], and convergence time can be reduced by using parallel learning agents [8], where each agent learns from its experience of visited states, and learns the value of unvisited states from other agents, and a Q-table can be shared among all the agents. A high number of states and or actions would lead to complex Q-tables with millions of cells and consume large amount of memory. Exploring all the states to generate a Q-table can also be time consuming [27]. To solve the challenges with large state and action space, techniques such as tile coding and function approximation can be used [12, 29, 36]. Other approaches to reducing the problem of dimensionality include aggregation, where multiple states or actions are aggregated to a smaller number of abstracted categories [8, 10, 19].

In this paper we develop an RL based controller to solve the VM migration problem and combine Q-Learning with an aggregated state action space to address the curse of dimensionality in Q-learning. To speed up RL convergence, we utilise parallel learning agents that learn from a shared collective experience of all agents. We develop a reward function that focuses on learning a policy to reduce SLA violations, and balance this with energy consumption.

## 5 Hybrid Architecture

Cloud Providers (CPs) build and operate large scale data centres that contain numerous computing resources, that are typically virtualised and require a level of orchestration of the shared resources, which is a challenging issue [3]. We classify the management process into two dimensions, *Management Algorithm (MA)* and *Management Framework (MF)*. The MA is responsible for deciding how workloads are assigned to infrastructure resources, while the MF enables the MA to execute by providing common functionality, such as hierarchy level management and aggregation of metrics between nodes. The combined functionality results in workloads executing on infrastructure nodes. In our previous work we detail a hybrid MF [24], which we summarise here. A new RL based MA is the focus of this paper.

The Management Framework (MF) provides common utilities that enable the MA to execute, including a mechanism to propagate node state, and a decision engine architecture that may be centralized, hierarchical or decentralized. The architecture of our previous work [24] consists of hybrid hierarchical decentralized controllers operating at different scopes. On the lowest level, every node in the infrastructure contains a Node Controller (NC), which dynamically adjusts resource configurations to satisfy VM demands on each node. A collection of NCs form a cluster of nodes. Each NC cooperates with a Lead Node (LN), which is a higher level controller for all the NCs within a given cluster. Unique to our proposal, the NCs within each cluster are divided into logical groups, called overlays, where a NC cooperates with other NCs within the same overlay. Each NC exists in only one overlay and in one cluster. Once again unique to our approach, the LN operates as a normal node within the infrastructure in addition to its management responsibility towards the cluster. At the highest level, the Data Centre Controller (DC) manages the controllers one level below it.

Our scalable hybrid architecture, SHDF, attempts to service resource requests at the lowest local level possible, in order to reduce the overhead of servicing the request [39] and to reduce the performance impact of migrating VMs across cluster boundaries [4].

## 5.1 Controller Functionality - VM Migration

When a node cannot satisfy the demands of the VMs it hosts, it starts an escalation process that aims to resolve the request at the lowest possible level. The MA running on the stressed node and the LN cooperate to resolve the escalated VM migration, by using our provided framework mechanisms.

The process starts within the NC's overlay by sending a request to other nodes within the same overlay (Step 1), shown in Fig. 3, by using the accumulated metrics of other nodes. If a target node is available and accepts the migration request, then the migration process completes for this cycle. If the selected target node does not accept, other nodes within the overlay are attempted until no further options are available within the overlay. If a target is not available within the overlay, the stressed NC escalates the VM migration request to the LN (Step 2), and the MA running on the LN can query the cluster records from all the overlays, which have state data from all nodes in the cluster, to find a suitable node to house an escalated VM. If the LN locates a target within the cluster, the migration request is forwarded (Step 3). If the LN cannot find a suitable target for the migration within the cluster, it will use its knowledge of other available clusters, through participation in the LNs overlays, to forward the migration request to another LN (Step 4). This target LN will repeat the process performed by the forwarding LN, and attempt nodes within this cluster (Step 5). If a suitable target is found the process competes. If the LN in Step 5 cannot find a target, the request is rejected back to the forwarding LN, which will attempt other LNs in its overlay. If a suitable target is found through another LN, the process completes. If this fails to find a suitable target, the request is escalated to the DC Controllers (Step 6). The DC has a view of the entire infrastructure and can forward the request to other LNs (Step 7), which the original LN does not cooperate with. This recipient LN will repeat the process carried out by other LNs in the

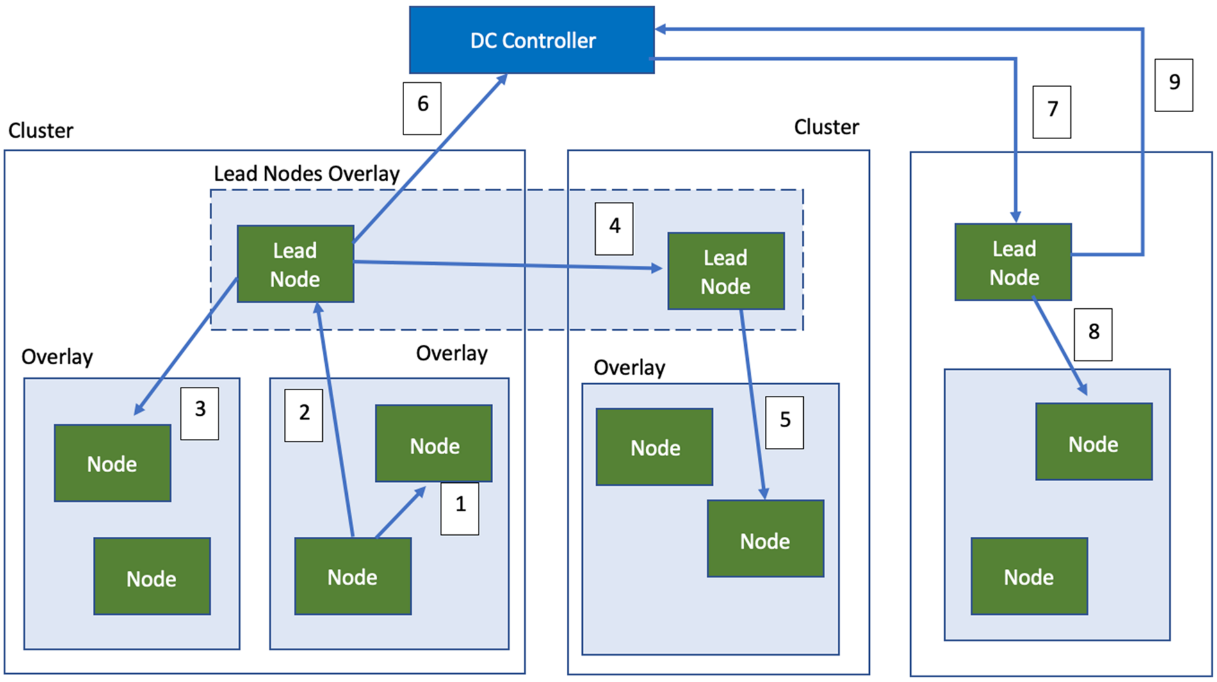


**Fig. 3** VM migration & escalation process

escalation chain (Step 8), and if a target is found the process completes. If a target is not found, the recent LN will reject the request back to the DC controller (Step 9). The DC controller will attempt other LNs, which have not been already tired, until a target is found or all LNs have been attempted.

If a target is not found then the infrastructure is highly stressed and the request is rejected back to the original escalating NC. In each of these escalation phases, the MA uses data from the data dissemination to decide on the list of targets to forward a request to. As requests progress through the escalation process, they are assigned an increasing priority, which can be used by the MA in the decision making process. For example, the MA may choose to prioritise finding a host for an escalated VM compared to a new VM placement.

### 5.2 Controller Functionality - Consolidation

At periodic time intervals and changes in utilisation, each of the management controllers and LNs can invoke a consolidation process where the MA can examine the state of the infrastructure and for every node under its management, decide to migrate some VMs from a node, migrate all VMs off a node and switch the node off or no change.

The advantage of SHDF is it allows the nodes to primarily operate in a distributed manner for time sensitive operations such as VM migrations, which could improve SLA violation metrics. In this paper, we focus on VM migration to achieve QoS metrics, and while RL consolidation is out of scope, we use a simple heuristic to perform regular consolidation.

## 6 Proposed Reinforcement Learning Management Algorithm

In our previous work [24], we have shown management algorithms (MAs) are widely covered in the literature, and drive the decision making process in cloud systems adaptation. The MA assigns resources in the infrastructure and regularly assesses the satisfaction of such assignments in achieving a given Service Level Agreement (SLA). The frequency of this assessment is influenced by the time complexity of the algorithm; the lower the complexity, the more frequently the algorithm can be executed. The approaches in the literature tend to be threshold-based [23], however, shared cloud environments have a significant uncertainty, and it is beneficial for the MA to be able to update the parameters of the decision making process to cope with a changing environment. With the promising results of applying RL in cloud resource management [8, 10, 19], we propose a RL based approach for VM migration, which builds on the MF from our earlier work, specifically to satisfy QoS metrics such as SLA. We build on SHDF by adding multiple modules that implement RL based agents at different levels of the SHDF hierarchy. On both the NC and LN, we add *monitoring, classification and learning modules* that provide the RL capability. The NC and LN perform their roles and escalation process on the hybrid architecture, as shown in Fig. 3. The new modules and their operation are shown in Fig. 4 and discussed below:

- A *Monitoring Module* tracks VM response times and is used as input by other modules to manage the node. The module additionally tracks the outcome of reinforcement learning actions.
- A *Classification module* assesses the state of a node and the VMs running on it, by using input

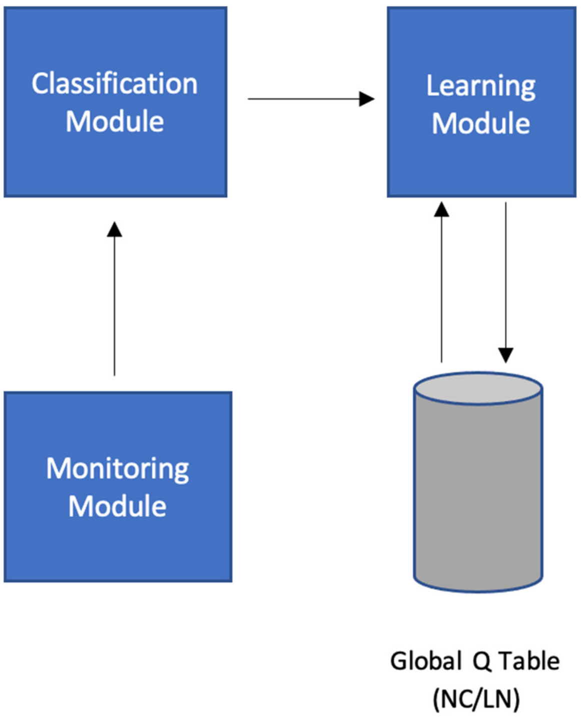


**Fig. 4** Proposed RL decision making steps

from the Monitoring node. The module decides if a VM is stressed.

– A *Learning Module* uses the other modules as input to carry out decision making. When a VM is classed as stressed, the module determines the available actions and runs the Q-learning algorithm to decide the action to take. The module additionally executes the action and invokes the monitoring module to determine the outcome of the action, calculates the reward for each taken action and updates the Q-table.

Algorithm 1 shows the operations in the RL agent. Each agent initialises value estimates to 0, and in each decision making cycle, the agent classifies the current state of the node, which identifies stressed VMs. Based on the current state of the node, the learning module determines all possible options by calling *getPossibleActions*. This uses the current overlay state in *state.targetCPUgroup[i]* to determine the available target groups of CPU utilisation from 0 to 90%. From the available options, the learning module will choose an action using an $\epsilon$-greedy policy with respect to Q. This policy ensures that not all the agent's actions are greedy with respect to Q, and the agent will sometimes choose a random action, which enables a tradeoff between exploration and exploitation. Based on the chosen action, the learning module will then execute the action.

To speed up learning and speed of convergence in the Q-learning, we employ parallel learning [6]. Each agent learns from its experience of visited states, and learns the value of unvisited states from cooperating nodes in the SHDF architecture. A Q-table is shared among all the agents for each level in the SHDF hierarchy, and thus there is a shared Q-table for all NCs and a separate shared Q-table for all LNs. In addition to speeding up the learning and convergence, this approach enables a unique cooperation between NCs and LNs. When a NC escalates a migration to the LN in a given state, the reward from this action is tracked as part of the learning process.

**Algorithm 1** RL@NC.

```
1:  procedure REGULARCHECK
2:      state ← classifyState(VM.getMonitoringData)
3:      actions ← state.getPossibleActions
4:      stateAction ← LearningModule.chooseAction(actions)
5:      switch stateAction do
6:          case powerNode
7:              result ← powerNewNode(vmToMigrate)
8:          case migrate
9:              result ← findOverlayNode(vmToMigrate, targetCPUWindow)
10:         case escalateToLead
11:             escalateToLead
12:         case noAction
13:             nop
14: procedure GETPOSSIBLEACTIONS(state)
15:     possibleActions ← null
16:     for targetCPUgroup[i] do
17:         if state.targetCPUgroup[i] > 0 then
18:             possibleActions.add ← targetCPUgroup[i]
19:     if offNodes > 0 then
20:         possibleActions.add ← powerOnNode
21:     possibleActions.add ← escalateToLead
22:
23: procedure CHOOSEACTION(possibleActions)
24:     if random < 1 −ϵ then
25:         action ← actionWithMaxQAtState(state, possibleActions)
26:     else
27:         action ← randomAction(possibleActions)
28:
```

## 6.1 RL State

The representation of the state is key to the RL decision making process. To overcome the state space dimensionality challenge with RL, we use a reduction approach and aggregate VMs to two states: Normal and Stressed. Response time has been used as a measure for application performance [18]. To account for variation in response time during the lifetime of an application, we use an approach similar to [43]. We apply linear regression on collected response time during each monitoring epoch, which by default is every two minutes. The classification module will deem a VM stressed when the 95th percentile of response time, during a monitoring period, is above a defined SLA threshold that by default is 500ms. We categorise the state of VMs as *Normal* when the 95th percentile of the response time is below the defined SLA level. The classification of state occurs during the regular node check, shown in line 2 of Algorithm 1.

## 6.2 RL Actions

Each node contains an RL agent, which carries out decision making. The RL agent carries out actions to achieve QoS metrics and balance this with energy consumption. To perform migration when a VM is stressed, the agent needs to identify a new target node to host the VM. The hybrid architecture provides an expanding set of options to migrate a VM, starting within the overlay where the VM resides and then onto the cluster and the rest of the data centre, a cooperation facilitated through the LN.

RL actions are contextual to the current state, and the agent ensures actions are valid by filtering non applicable actions, as shown in the getPossibleActions method in line 3 of Algorithm 1. For example, when the VM is in *Normal* state, no migration actions are available to the RL agent.

The goal for the RL agents is to find the actions that reduce SLA violations (maximise reward) when an agent enters a given state, shown in method *chooseAction* in line 4 of Algorithm 1. During the decision making process, the agent chooses between powering on a new node to house the migrating VM (line 6), migrating the VM to another target node within the same overlay (line 8), escalating to the Lead Node (line 10) and taking no action (line 12). In the migrate within overlay case, the RL agent needs to choose a target node to send a stressed VM to. However, a large data centre will have many thousands of nodes, and tracking an action reward for each individual target node will lead to a large set of actions and a large Q-table. To solve this, we simplify the RL action space and use a reduction mechanism that groups target nodes based on their CPU utilisation. By default we use 10 groups, 10% each using (2), which creates target groups from 0 to 9. For example, action group1 means migrate the stressed VM to a node with an average CPU utilisation of 10% to 19%. Selecting action group6 means migrate to a node with CPU utilisation of 60% to 69%. Once an action is selected, we use a greedy policy to select the first available node that fits the action group. For example, an action of group6, will result in the first available node that meets the requirements of the VM and has an average utilisation between 60% and 69% to become selected as the target node. Based on CPU utilisation, we identify the target groups available and add these as options for the RL agent to choose from (lines 16 to 18). We additionally account for the number of available switched off nodes and add these as an option to the RL agent to switch on (lines 19 to 20). In most cases the agent chooses an action, from the available options, that maximises future reward (line 25). With a small probability of $\epsilon$, the agent will choose to explore and not exploit, by randomly selecting an available action (line 27).

$$targetGroup = \frac{avgCpuUtilization(node)}{actionGroups} \tag{2}$$

The NC can select actions to migrate a stressed VM within the overlay, as well escalate to a LN. We modify the original hybrid architecture, where escalation to the LN was only available when no target nodes were available in the overlay, to being able to escalate at any point using a RL action (line 12). The RL based migration and escalation process is shown in Fig. 5. A LN deals with escalation of VM migrations from a NC, and performs RL actions within the scope of a whole cluster. Table 2 shows the RL actions carried out by a NC and Table 3 shows the LN actions. Similar to the original hybrid architecture, the process starts within the NC's overlay, where a NC uses the RL modules to make a migration. The difference in this new version, is the NC can choose to migrate within the overlay (Step 1) or make an escalation to the LN (Step 2), based on the RL agent and the actions from Table 2. If an escalation to the LN is chosen, the RL agent running on the LN will classify the cluster state and choose an action from Table 3, which can be within the cluster (Step 3) or a migration outside the cluster (Step 4). Once the action is executed, both the LN and NC RL agents receive a reward post action completion. The NC receives a reward for the escalation action (Step 5) and the LN for the choice of action within the cluster or outside it (Step 6).

Consolidation using RL is outside the scope of this work, and we use a simple heuristic to perform regular consolidation at the cluster level. The heuristic, based on [31], classifies all the nodes at the cluster level as *partially utilised, under utilised or empty*. All under utilised nodes become candidates for migrating VMs away from to other partly utilised nodes. When the

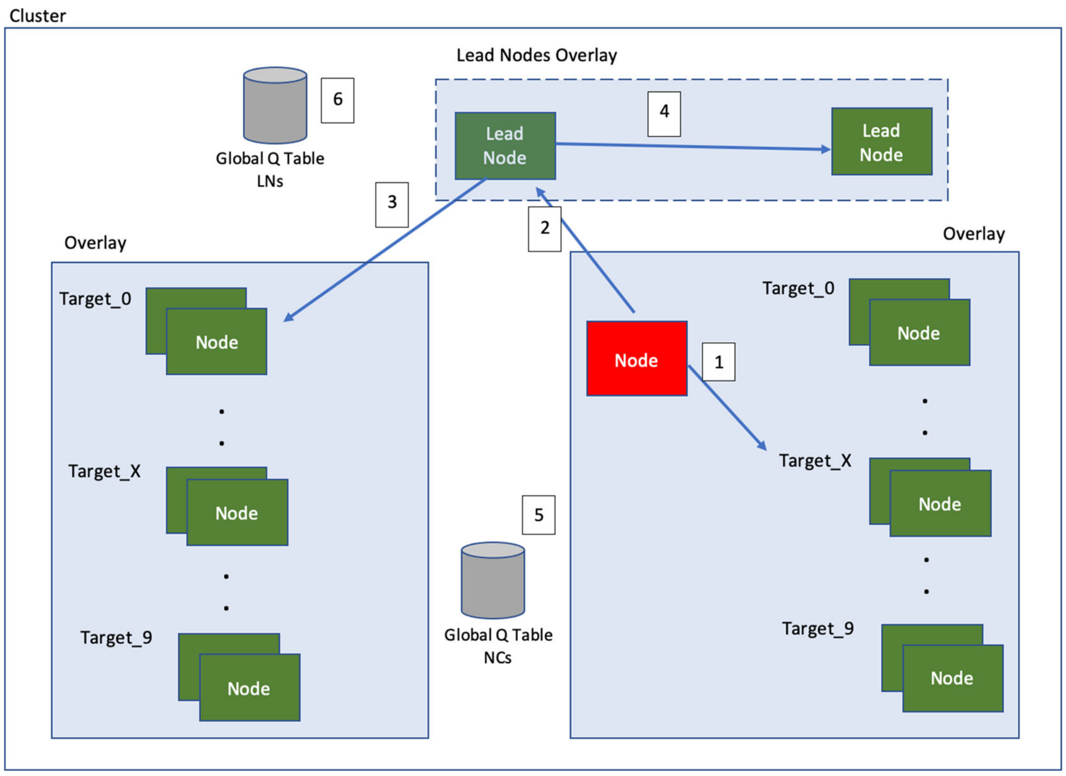


**Fig. 5** RL migration & escalation process

last VM is migrated away from a node and it becomes empty, the node is switched off.

**Algorithm 2** Update Q-table.

```
1: procedure UPDATEQ(action)
2:     s' ← classifyState(VM.getMonitoringData)
3:     responseTime ← monitoringModule.getRT(vm)
4:     reward ← calculateReward(responseTime)
5:     $Q(s, a) \leftarrow Q(s, a) + \alpha[reward + \gamma Q\_max(s', a) - Q(s, a)]$
```

## 6.3 Reward

The goal for RL is to maximise rewards through incrementally mapping states to actions. The monitoring and classification modules regularly monitor a node and capture the state, which enables a subsequent action from the learning module. After an action is executed, there is a waiting period, which is action dependent. Boot up actions take a defined amount of time, typically set to 30 seconds, while migrations take a length of time that is dependent on the amount of memory used by the VM. After the action wait time, the monitoring module will calculate a reward for the action using (3), and is shown in Algorithm 2. The update starts by classifying the current state of a VM (line 2), and calculating the archived reward based on (3) in line 4. The Q-learning update is then applied based on (1) in line 5. As our goal is to reduce SLA violations, and balance this with energy consumption, the reward function should reflect VM performance (yield) and resource utilisation (cost), and punish actions that degrade VM performance or significantly increase cost. Our reward function is shown in (3), where yield is the gain in a QoS metric, and cost is the increase in energy consumption, represented by the delta in CPU utilisation. As nodes consume up to 70% of their full utilisation in idle state [13], the *Power on node* action incurs a *penalty* of 0.7. Other actions do not carry a penalty.

$$Reward = yield - cost - penalty, \qquad (3)$$

**Table 2** NC RL actions

| Action | Description |
|---|---|
| Migrate [0,1,2,3,4,5,6,7,8,9] inside overlay | Migrate to target node with utilisation [1 to 9%, 10 to 19%, 20 to 29%, 30 to 39%, 40 to 49%, 50 to 59%, 60 to 69%, 70 to 79%, 80 to 89%, 90 to 99%] |
| Migrate Outside Overlay | Escalate migration to LN |
| Power Node inside overlay | Power on node & migrate |
| Do nothing | Do nothing |

where $yield \in [-1,1]$ decaying to $-1$ for the worst QoS and 1 when QoS is met. Cost is the change in CPU utilisation where $cost \in [0,1]$. We use *Response Time* as our key QoS metric. Response Time ($rt$) represents the time it takes to execute a request to an application running inside a hosted VM, and reflects the CPU resources that are assigned to the VM. The monitoring module captures response time in the 95th percentile. When a VM is moved to a new target node, it will likely experience a change in $rt$; migration may also impact other VMs running on the target node. To capture this, our yield of $rt$ is calculated based on (4), where $m$ is the number of all VMs running on the target node.

$$yield = \sum_{j=1}^{m} y(rt_j)\,. \tag{4}$$

For each VM, when $rt$ is a value below the $TargetRT$ and therefore satisfying SLA, the reward is 1. When $rt$ is above the $TargetRT$ for the VM, the function will punish actions that cause SLA violations, as shown in (5).

$$y\,(rt) = \begin{cases} TargetRT - rt & , rt > TargetRT, \\ 1 & rt < TargetRT \end{cases} \tag{5}$$

*Energy Consumption* is the second component used by the reward function, and captures the energy utilisation cost of the action, based on the CPU utilisation of the target node before and after the action. This value helps the learnt policy to move towards actions that balance meeting SLAs with energy consumption, and is shown in (6).

$$cost = postActionUtilisation - preActionUtilisation \tag{6}$$

Each NC and LN carries out an action, receives a reward for the action, and updates the shared global NC and LN Q-tables respectively, as shown in Algorithm 2.

## 7 Experiment Setup and Evaluation

In this section, we evaluate different approaches and compare our RL based Management Algorithm (MA) with multiple systems from literature, on both the MA and Management Framework (MF) dimensions. We choose to compare to a heuristic [7] MA that uses a Modified Best Fit Decreasing approach to migrate a VM, which has been shown to be effective in practice. For MF, we choose the hybrid architecture from our earlier work [24] and a centralized MF, due to the popularity of centralized MFs in the literature, and we combine MFs and MAs with varying workloads and VM configurations. Thus, our evaluation set is:

– RL MA combined with Hybrid MF

**Table 3** LN RL actions

| Action | Description |
|---|---|
| Migrate [0,1,2,3,4,5,6,7,8,9] inside cluster | Migrate to target node with utilisation [1 to 9%, 10 to 19%, 20 to 29%, 30 to 39%, 40 to 49%, 50 to 59%, 60 to 69%, 70 to 79%, 80 to 89%, 90 to 99%] |
| Migrate Outside cluster | Escalate migration to data centre controller |
| Power Node inside cluster | Power on node & migrate |

- Heuristic MA combined with Hybrid MF
- RL MA combined with Centralized MF
- Heuristic MA combined Centralized MF

This combination enables us to evaluate the performance of the proposed RL MA, by comparing to a Heuristic MA, and investigate benefits that can be realised from combining the RL MA with the Hybrid MF.

We used simulation to facilitate rapid development of experiments of large data centres. We selected DCSim [57] because of its extensibility and existing implementation of a centralized architecture, allowing us to create baseline comparators for our proposed RL approach. We instantiate SHDF with three levels of controllers, running on the root of the data centre (DC Controller), the cluster manager (LN) and executing nodes (NC).

### 7.1 Simulator Setup

DCSim allows a VM to use more CPU than reserved, up to an amount that does not impact other VMs. Like [31], for the heuristic based approaches we use a CPU utilisation thresholds of 90% for high, indicating stress level, and we use 60% for low, indicating low utilisation.

In DCSim, an application is modelled as an interactive multi-tiered web application. Each application has a specified client think time, a workload component and a request service time, which is the amount of time required to process each incoming request. The workload defines the current number of clients connected to the application, which can change at discrete points in the simulation based on a *trace* file. The resource requirements are defined as its resource size, which is the expected amount of CPU, memory, bandwidth and storage. DCSim treats bandwidth and storage as fixed requirements, however, CPU requirements can be varied across the simulation based on the VM demands. DCSim applies a cost to VM migration including the time taken for migration, as a function of memory consumed by a VM, and factors in the bandwidth required for the VM migration on the hosting node. Additionally, the boot time of a switched off node has an elapsed time cost. The time taken to switch on a node for migration is reflected in the time period the VM is in a stressed state, and therefore the SLA achieved by a VM. Due to the complexities of building accurate power models, we focus our investigation on scalability metrics.

### 7.2 Workload and SLA Violations

We run the experiments at a load that requires more than 70% of the CPU resources of active nodes. Each simulated application contains a workload trace based on the number of incoming requests to web servers from publicly available traces; we used the following traces included with DCSim: Google 1, Google 3, EPA (Environment Protection Agency) and Clarknet. Each workload has an average normalized load of: 0.74, 0.31, 0.24 for Google 1, Google 3, Clarknet and EPA respectively. Figure 6 shows the normalized shape of the workload requests for each of these traces. DCSim divides traces into equal length segments, and total the number of requests in each interval. The values of each interval are then normalized to [0, 1], with 0 being zero requests, and 1 being the maximum number of requests received in an interval. These are then scaled to match the VM being used; for example if a VM uses one core at 2GHz, the normalized trace is scaled by 2000. We create VMs with different cores and RAM configurations, as shown in Table 4. Each experiment is run to simulate 24 hours, and when there is not enough trace data for a an experiment duration, we loop to the beginning of the trace.

### 7.3 Modelling the Impact of Decision Making

DCSim [57] applies a migration cost once a VM is selected for migration, by adding additional time to complete the migration based on the amount of memory used by the VM. However, DCSim does not account for the time it takes to execute the decision making process, or the impact of such time. The length of the decision making process impacts stressed nodes by increasing the amount of time the node stays in a stressed state. In a centralized architecture, all nodes are used as input into the decision making process. Therefore, the execution of decision making could get progressively higher, as the number of managed nodes increases.

To capture the cost of the decision making, we extend DCSim to measure the amount of time during decision making, and add this time to the VM migra-

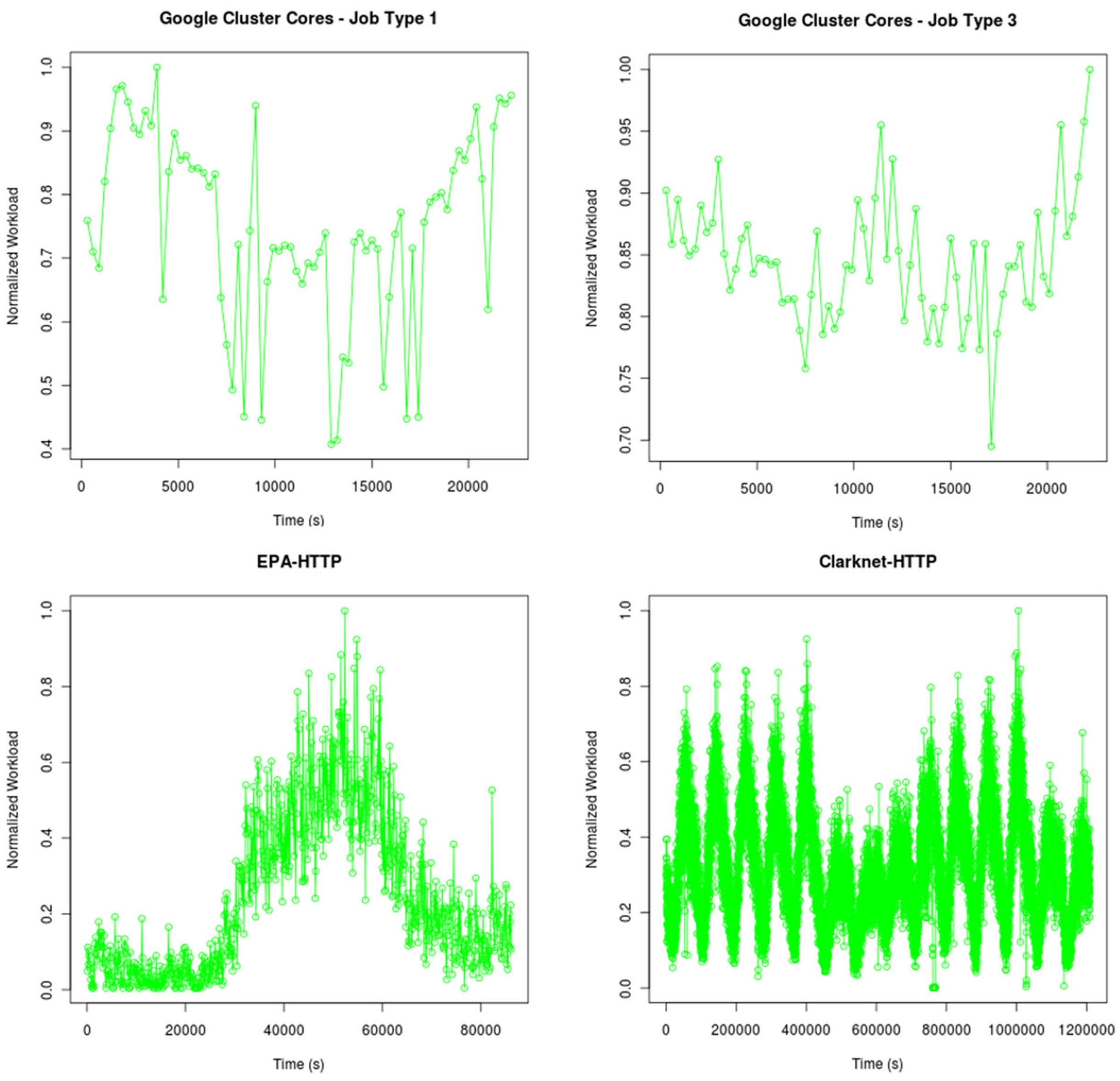


**Fig. 6** Original normalized traces used [57]

tion duration. As the decision making execution time varies based on the MA and the hardware it is running on, we add a configurable scaling factor that can be applied to the measured execution time.

## 7.4 Data Centre

Our experiments use nodes modelled as ProLiant DL380 G5 Quad Core [1], with 2 dual-core 3GHz CPUs and 16GB of memory. The number of nodes used is specified in each of the experiments, with a minimum of 1,000 nodes. We assume that the data centre supports live VM migration, as this technique is currently supported by most major hypervisor technologies, such VMware [2] or Xen [14].

To minimise the number of variables in our experiments, we chose to keep a homogeneous infrastructure, with the same specification for all of the nodes. The various parameters used in our evaluation are outlined in Table 4.

**Table 4** Experiment configuration

| Config | Config options | Base config |
|---|---|---|
| VM Core (Mhz) | 1000,1300,2500 | round robin [1000,1300,2500] |
| Node Capacity (Mhz) | 3000 | 3000 |
| Number of Cores | 1,2 | round robin [1,2] |
| VM Memory (MBs) | 1024,2048 | round robin [1024,2048] |
| Workload | clarknet, EPA, Google 1, Google 3 | round robin [clarknet, EPA, Google 1, Google 3] |
| Number of Nodes | 1000, 2000,3000, 4000, 5000 | 1000 |
| Node stress check frequency | 2 minutes | 2 minutes |
| VM service time | 0.2 seconds | 0.2 seconds |
| RL parameters | $\alpha$=0.1, $\gamma$=0.7, $\epsilon$=0.1 | $\alpha$=0.1, $\gamma$=0.7, $\epsilon$=0.1 |
| Target Response Time | 0.5 | 0.5 |

## 7.5 Experiments

Our goal is to demonstrate improved QoS metrics for the RL MA, and we evaluate this by examining *SLA violations*. We evaluate our proposal under varying workloads and draw a comparison between our proposal and several related techniques.

Simulated applications are modelled as interactive web servers, running inside a VM, and an *SLA Violation* occurs when response time associated with the VM exceeds the *target response time*. We evaluate all architectures under varying scenarios to understand the impact on SLA violations. Initially, we evaluate a *mixed workload* scenario to represent the varying workloads deployed on data centres. To understand the impact of specific workloads, we then evaluate these individually. We additionally examine scalability and how the approaches cope with a varying arrival rate for new workloads.

### *7.5.1 No Adaptation*

In this experiment, we use a combined workload of all traces in a round robin approach, where each created VM uses a workload trace from all of the available traces. We use the centralized architecture with 1000 homogeneous nodes and no adaptation is invoked. When a node becomes stressed, it remains stressed for the reminder of the experiment, and exhibited 117045 instances of SLA violations.

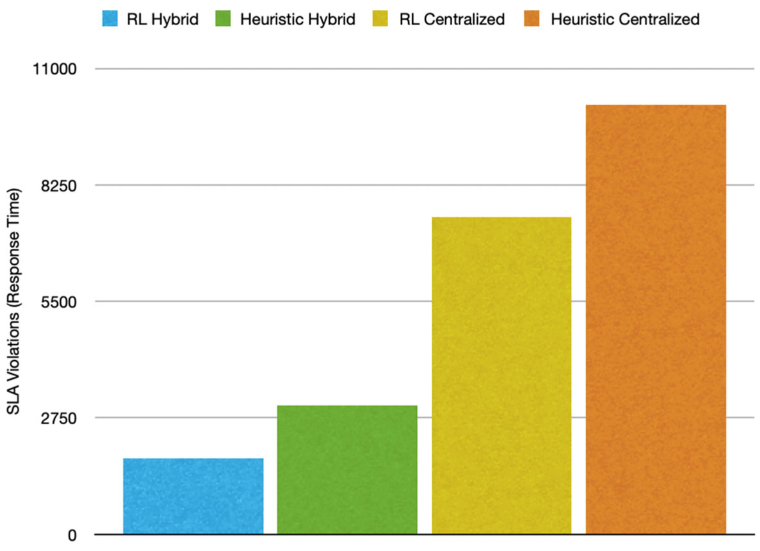


**Fig. 7** Mixed workload results: Number of SLA Violations response time > 0.5s

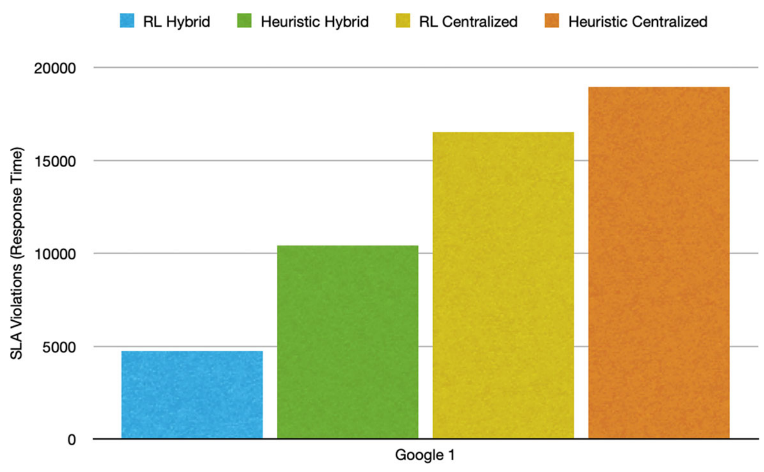


**Fig. 8** Google 1 workload results - Number of SLA Violations response time > 0.5s

*7.5.2 Mixed Workload Assessment*

In this experiment, we use a combined workload of all traces in a round robin approach, where each created VM uses a different workload trace, in a deterministic order. The mixed workload simulates different workloads that could be experienced in a data centre. Workload arrival rate is the frequency that new VM placement requests arrive at the data centre. For this experiment we use an arrival rate of 180 new VMs per hour, with each VM running for 10 hours before shutting down. The experiment simulates 24 hours of elapsed time. We use 1000 nodes and the base configuration in Table 4. The SLA violation results are shown on Fig. 7.

The hybrid architecture achieved lower SLA violation compared to centralized, and the RL hybrid outperformed by 69.9%, 320.0% and 468.3% against the heuristic hybrid, RL centralized and heuristic centralized, respectively. This is due to the RL approach of discovering target allocations that achieve the desired VM response time, and thus reduce SLA violations. The reward function drives behaviour of RL to choose target nodes that are switched on and have lower existing CPU utilisation. While the centralized architecture benefited from using RL, the hybrid architecture benefited more due to its autonomous approach to decision making, which provides rapid decision making, combined with a target node selection that maintains VM response time, thus reducing SLA violations.

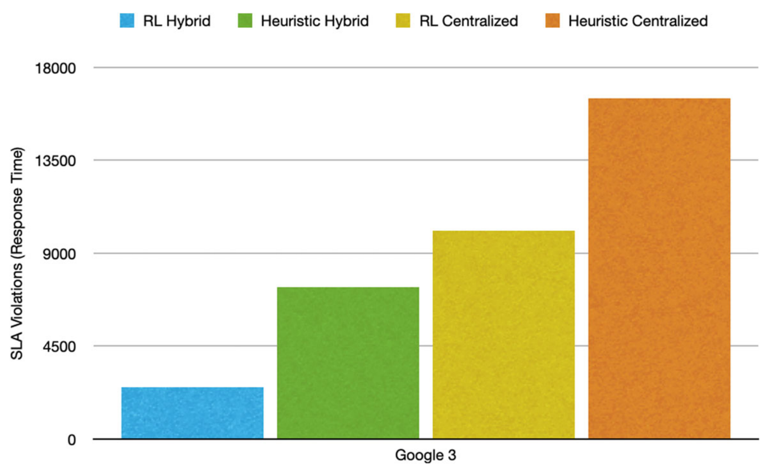


**Fig. 9** Google 3 workload results - Number of SLA Violations response time > 0.5s

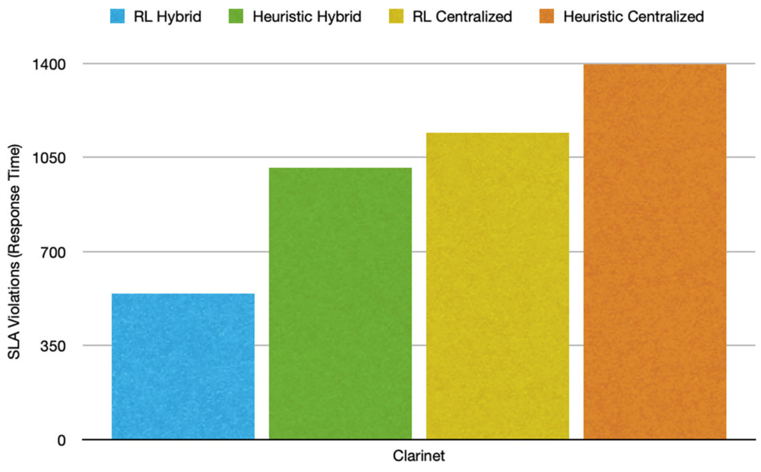


**Fig. 10** Clarknet workload results - Number of SLA Violations response time > 0.5s

Compared to the *No adaptation* experiment, all approaches unsurprisingly reduced SLA violations.

## 7.6 Workload Impact

To investigate the impact of workload further, we use the individual workloads from the previous experiment and evaluate them individually. For this experiment we use an arrival rate of 180 new VMs per hour, with each VM running for 10 hours before shutting down. The experiment simulated 24 hours of elapsed time. We use the base configuration in Table 4.

The results for the Google 1 workload are shown in Fig. 8, and show the effect on SLA violations. This workload has an average of 0.74 normalized load and thus high stress. The RL hybrid approach outperformed other approaches by 119.4%, 248.4% and 299.7% against the heuristic hybrid, RL centralized and heuristic centralized respectively. The RL hybrid approach continues to perform well on this stressful workload, and shows further improvement in this workload against the heuristic hybrid; 119% versus 69% in the mixed workload, indicating the hybrid architecture benefits from the RL learnt policy, which favours target nodes that achieve target response time, while the heuristic approach does not take feedback signal on target node selection.

The results for the Google 3 workload are shown in Fig. 9, and show the effect on SLA violations. This workload has an average of .83 normalized load and is the highest workload stress we have evaluated. The hybrid approach outperformed other approaches by 192.0%, 300.6% and 555.0% against the heuristic hybrid, RL centralized and heuristic centralized

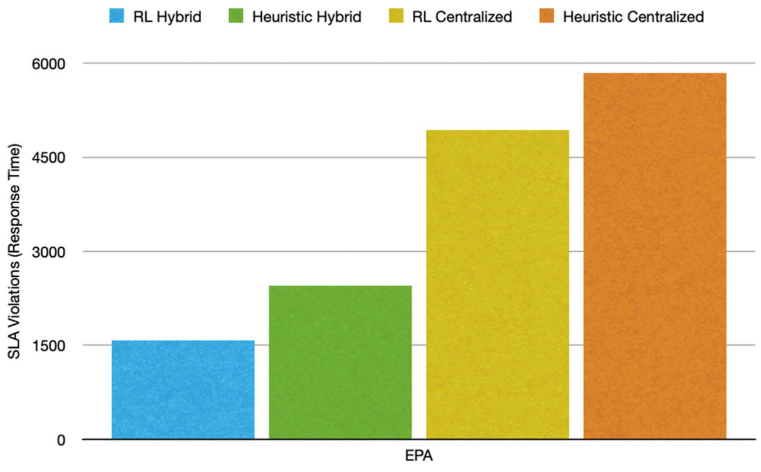
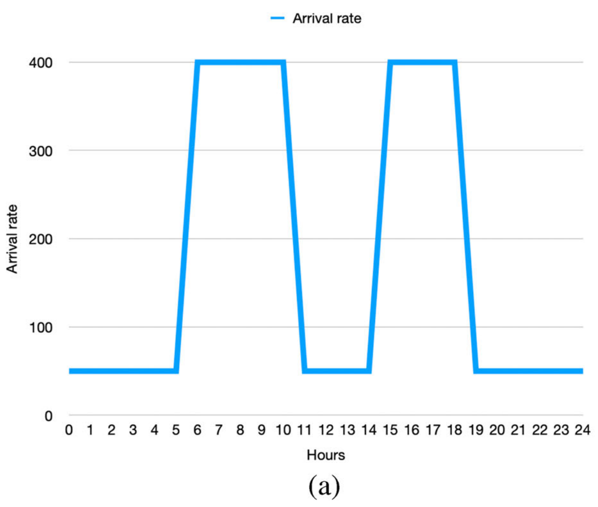


**Fig. 11** EPA workload results - Number of SLA Violations response time > 0.5s



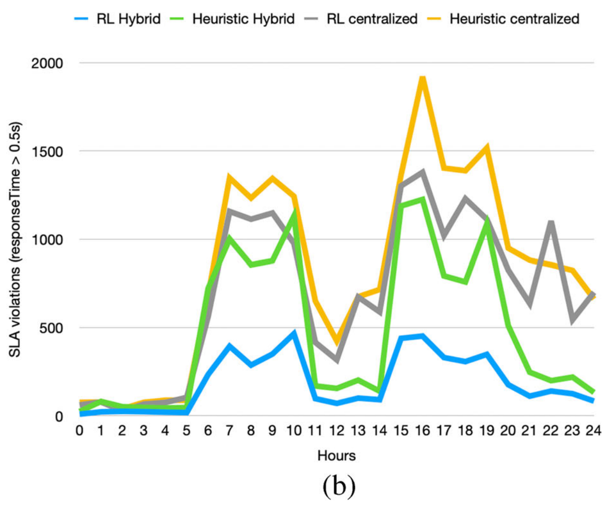


**Fig. 12** High dynamic workload: a)Arrival rate patterns, b) SLA violations for dynamic arrival rate

respectively. The RL hybrid approach performs even better on this workload, compared to the Google 1 workload, indicating that more stressful workloads benefit form the learned RL policy, which favours target node selection that meets target response time.

The results for the Clarknet workload are shown in Fig. 10, and show the effect on SLA violations. This workload has an average of .31 normalized load and is a lower stress compared to the Google workloads. The hybrid approach outperformed other approaches by 85.7%, 109.9% and 156.6% against the heuristic hybrid, RL centralized and heuristic centralized, respectively. These are lower improvements, indicating higher stress workload benefit more from the RL hybrid approach.

The results for the EPA workload are shown in Fig. 11, and show the effect on SLA violations. This workload has an average of .24 normalized load and is the least stress workload we have evaluated. The hybrid approach outperformed other approaches

**Fig. 13** Active Nodes in RL Hybrid versus Heuristic Hybrid

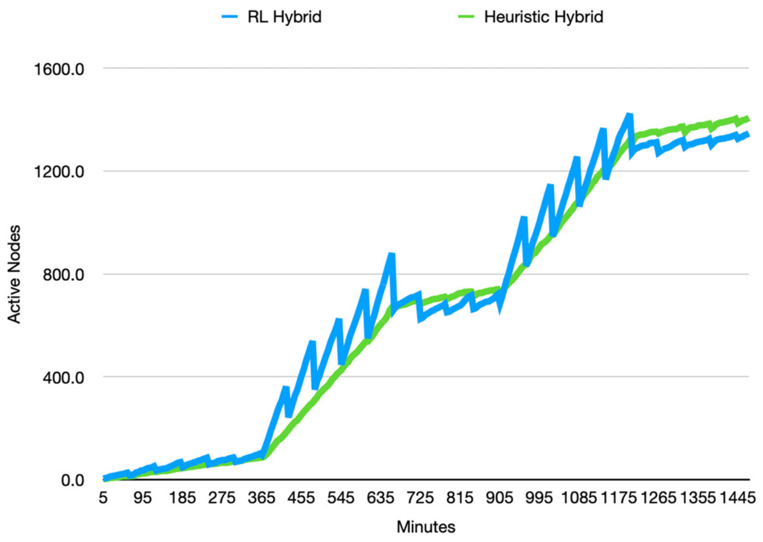


**Table 5** Scalability of the different approaches - SLA Violations

| Number of nodes | RL hybrid | Heuristic hybrid | RL centralized | Heuristic centralized |
|---|---|---|---|---|
| 1000 | 2518 | 7353 | 10257 | 13788 |
| 2000 | 5485 | 18817 | 22357 | 29262 |
| 3000 | 8537 | 29784 | 34161 | 44044 |
| 4000 | 11051 | 43633 | 46882 | 59026 |
| 5000 | 17771 | 56678 | 58730 | 75208 |

by 54.9%, 212.1% and 269.2% against the heuristic hybrid, RL centralized and heuristic centralized, respectively. These are lower improvements, indicating higher stress workloads benefit more from the RL hybrid approach.

## 7.7 Dynamic Workload

To evaluate how the different approaches cope with multiple high arrival rates, this experiment involves several sharp increases in arrival rate, by 400 VMs per hour each time. High arrival rate for VMs causes more VMs to be placed in the infrastructure, and in turn more VMs that can exhibit SLA violations. We examine the scenario in Fig. 12a. There is a small and steady flow of new VM creation requests, followed by two sharp increases in number of VM creation requests that persist for several hours, followed by a return to the previous small steady number of VM creation requests. Each VM runs for 1 hour before shutting down, and we use the Google 3 trace. The experiment simulates 24 hours of elapsed time, and we use the base configuration in Table 4. The high arrival rate of VMs on the infrastructure, as each VM runs the Google 3 workload, increases the VMs that begin to experience a stress state where they do not deliver the requested CPU demand, and thus enter SLA violation. We hypothesised that this could be challenging for RL approaches, as they could learn suitable responses for arrival rates that are not sustained.

The results are shown in Fig. 12b, and show number of incurred SLA violations. The hybrid based approaches outperform the centralized approach, due to the rapid decision making process. The RL hybrid outperforms the heuristic hybrid by receiving a reward for powering on new nodes as the load increases,

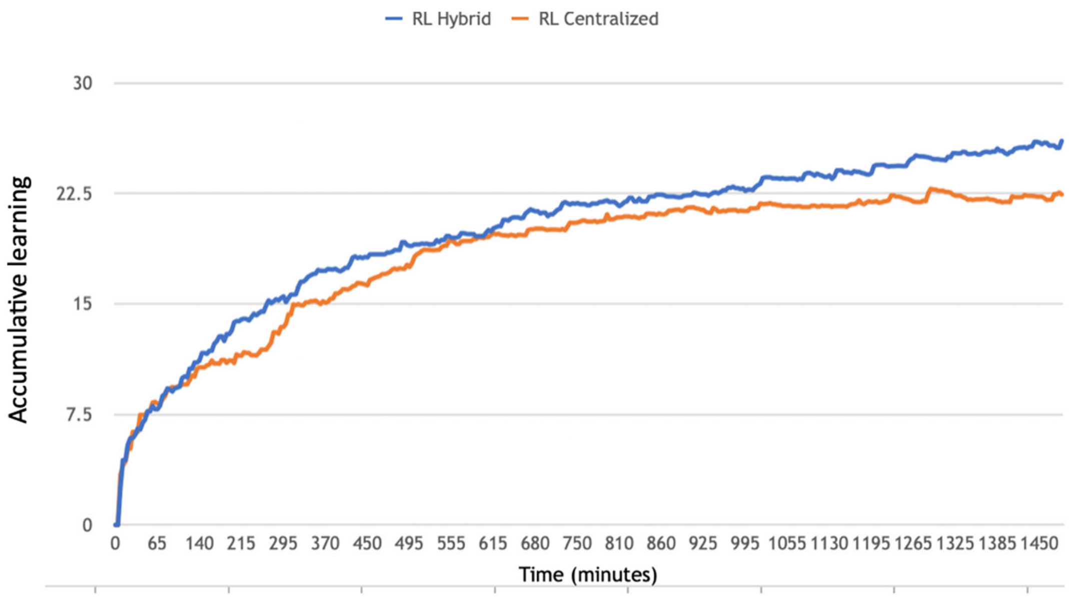


**Fig. 14** RL convergence: Hybrid versus Centralized

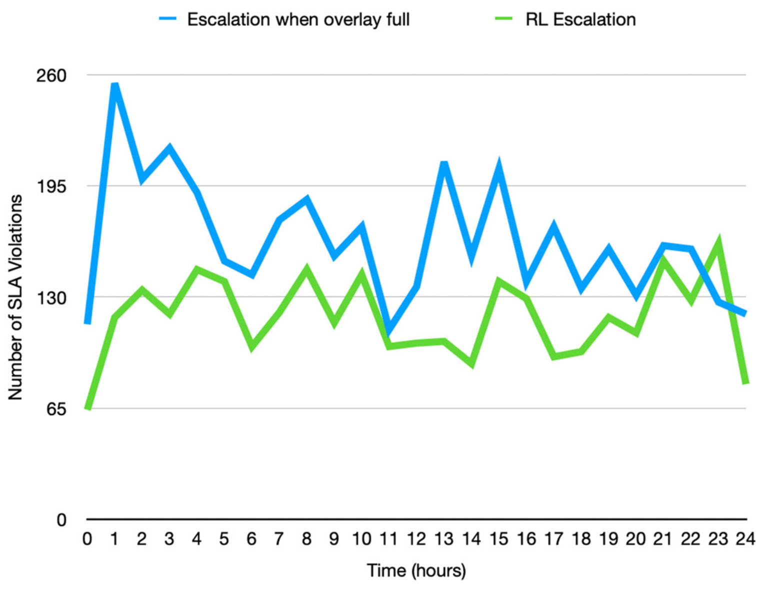


**Fig. 15** SLA violations: Hybrid Heuristic versus Hybrid RL with escalation

shown in Fig. 13. During each of the spikes at hours 5 and 14, the RL switched on more nodes versus steady new nodes in the heuristic hybrid. While the RL centralized has an improved target node allocation, compared to the heuristic centralized, it continued to suffer from the time taken for the decision making inherent in the centralized architecture.

### 7.8 Scalability

To evaluate how the approaches scale, we use the most stressful workload, Google 3, and maintain the stress ratio for each node, by increasing the load and the number of nodes in the data centre. Similar to other experiments, we simulate 24 hours and the results are shown in Table 5.

As the load ratio increases with more nodes, it is expected the number of SLA violations increases. The RL hybrid approach maintains its SLA performance as more nodes and VMs are added to the data centre, with lower SLA violations than all other approaches, on average 239.5%, 294.0% and 410.9% compared to the heuristic hybrid, RL centralized and heuristic centralized respectively.

### 7.9 Learning Performance

Convergence behaviour of an RL agent is a useful indicator to show whether the agent is learning an optimal policy. A preferable convergence shape is one where cumulative reward can gradually increase through time and converge to a high value [66]. While the RL hybrid and RL centralized use the same RL approach to monitor, classify and action node state, they have different decision making architectures. The RL hybrid performs parallel decentralized learning and the centralized has a single learning agent. The hybrid architecture has more rapid decision making and is able to execute more actions in a given time window, and is able to converge on a policy faster than the centralized approach. Figure 14 shows the cumulative value of RL actions for both approaches, from the Google 3 workload experiment. Initially there is low stress on the infrastructure, and the RL approach is making *No-operation* actions and both RL hybrid and RL centralized are receiving similar rewards. As the stress on the infrastructure increases, the RL hybrid is able to observe rewards from actions quicker than the RL centralized approach, and thus accumulate higher value during the experiment.

The unique cooperation between the RL agents running on NCs and LNs, enables escalation by the NC within one overlay to other overlays in the cluster. In this paper, we extended the original hybrid architecture, where escalation to the LN was only performed when no target nodes where available in the overlay, to NCs being able to escalate at any point using an RL action. This enables the RL agent to explore escalation to LN prior to the overlay becoming full. Figure 15 shows the reduction in SLA violations in the

RL escalation approach. By performing opportunistic escalation, the NC is reducing pressure on the overlay by leaving capacity in the overlay and thus reducing SLA violations.

## 8 Conclusion

RL has shown great promise in a range of control applications. Using RL in managing cloud infrastructures offers adaptability and advantages over heuristic based approaches, which historically have been threshold based and require significant domain and application knowledge to define threshold values. In this paper, we presented a RL management algorithm that reduces the state and action space and uses a unique multi level RL agent cooperation, between a NC and LN in hierarchical management, to further improve SLA violations performance. This RL management algorithm integrates well into a hybrid management framework, from our earlier work. We evaluated the performance of our approach using workload traces and simulation, and compared the results obtained with an established heuristic, demonstrating significant improvement to SLA violations and high scalability. Future areas of improvement include expanding the RL state space, and enabling the RL MA to learn to migrate VMs just before they become stressed. Our RL approach can also be extended to include a RL approach for initial VM placement and consolidation. Additionally, it would be valuable to validate the simulation results with experimentation on actual cloud infrastructure.

**Data Availability** The data traces used as input for this paper are part of the DCSim [57] simulator, and are available from Github: https://github.com/digs-uwo/dcsim/tree/master/traces. Most of the data generated by this research is contained in the results section of this paper, and the full datasets generated are available from the corresponding author on request.

**Conflict of Interest** The authors declare that they have no conflict of interest.

## References

1. Hpe proliant (2016). https://www.hpe.com
2. Vmware. (2016). http://www.vmware.com/
3. Ahmad, R.W., Gani, A., Hamid, S.H.A., Shiraz, M., Yousafzai, A., Xia, F.: A survey on virtual machine migration and server consolidation frameworks for cloud data centers. J. Netw. Comput. Appl. **52**, 11–25 (2015). https://doi.org/10.1016/j.jnca.2015.02.002
4. Aldhalaan, A., Menascé, D.A.: Autonomic Allocation of Communicating Virtual Machines in Hierarchical Cloud Data Centers. In: 2014 International Conference On Cloud and Autonomic Computing (ICCAC), pp. 161–171. 2014 International Conference On Cloud and Autonomic Computing (2014) (2014)
5. Arabnejad, H., Pahl, C., Jamshidi, P., Estrada, G.: A Comparison of Reinforcement Learning Techniques for Fuzzy Cloud Auto-Scaling. In: 2017 17Th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 64–73. https://doi.org/10.1109/CCGRID.2017.15 (2017)
6. Barrett, E., Howley, E., Duggan, J.: Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. Concurr. Comput. Pract. Exper. **25**(12), 1656–1674 (2013). https://doi.org/10.1002/cpe.2864
7. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput. Pract. Exper. **24**, 1397–1420 (2012)
8. Bibal Benifa, J.V., Dejey, D.: Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment. Mobile Netw. Appl. **24**(4), 1348–1363 (2019). https://doi.org/10.1007/s11036-018-0996-0
9. Bitsakos, C., Konstantinou, I., Koziris, N.: Derp: a Deep Reinforcement Learning Cloud System for Elastic Resource Provisioning. In: 2018 IEEE International Conference on Cloud Computing Technology and Science (Cloudcom), pp. 21–29. https://doi.org/10.1109/CloudCom2018.2018.00020 (2018)
10. Bu, X., Rao, J., Xu, C.Z.: Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. IEEE Trans. Parallel Distrib. Syst. **24**(4), 681–690 (2013). https://doi.org/10.1109/TPDS.2012.174
11. Calcavecchia, N.M., Caprarescu, B.A., Di Nitto, E., Dubois, D.J., Petcu, D.: Depas: a decentralized probabilistic algorithm for auto-scaling. Computing **94**(8), 701–730 (2012)
12. Chen, Z., Hu, J., Min, G.: Learning-Based Resource Allocation in Cloud Data Center Using Advantage Actor-Critic. In: ICC 2019 - 2019 IEEE International Conference on Communications (2019), pp. 1–6. https://doi.org/10.1109/ICC.2019.8761309 (2019)
13. Chowdhury, M.R., Mahmud, M.R., Rahman, R.M.: Implementation and performance analysis of various vm placement strategies in cloudsim. J. Cloud Comput. **4**(1), 20 (2015). https://doi.org/10.1186/s13677-015-0045-5
14. Citrix: Xen. (2016). http://www.xenserver.org
15. Duggan, M., Flesk, K., Duggan, J., Howley, E.: Barrett e.: a reinforcement learning approach for dynamic selection of virtual machines in cloud data centres. https://doi.org/10.1109/INTECH.2016.7845053 (2016)

16. Dulac-Arnold, G., Levine, N., Mankowitz, D.J., Li, J., Paduraru, C., Gowal, S., Hester, T.: Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning. https://doi.org/10.1007/s10994-021-05961-4 (2021)
17. Gahlawat, M., Sharma, P.: Survey of Virtual Machine Placement in Federated Clouds. In: 2014 IEEE International Advance Computing Conference (IACC), pp. 735–738. https://doi.org/10.1109/IAdCC.2014.6779415 (2014)
18. Ghanbari, H., Simmons, B., Litoiu, M., Barna, C., Iszlai, G.: Optimal autoscaling in a iaas cloud. In: Proceedings of the 9th International Conference on Autonomic Computing, ICAC '12, pp. 173–178. Association for Computing Machinery, New York (2012). https://doi.org/10.1145/2371536.2371567
19. Ghobaei-Arani, M., Jabbehdari, S., Pourmina, M.A.: An autonomic resource provisioning approach for service-based cloud applications: a hybrid approach. Fut. Gener. Comput. Syst. **78**, 191–210 (2018). https://doi.org/10.1016/j.future.2017.02.022
20. Gholipour, N., Arianyan, E., Buyya, R.: A novel energy-aware resource management technique using joint vm and container consolidation approach for green computing in cloud data centers. Simul. Model. Pract. Theory **104**, 102127 (2020). https://doi.org/10.1016/j.simpat.2020.102127
21. Guo, W., Tian, W., Ye, Y., Xu, L., Wu, K.: Cloud resource scheduling with deep reinforcement learning and imitation learning. IEEE Internet Things J. **8**(5), 3576–3586 (2021). https://doi.org/10.1109/JIOT.2020.3025015
22. Gupta, M.K., Amgoth, T.: Resource-aware virtual machine placement algorithm for iaas cloud. J. Supercomput. **74**(1), 122–140 (2018). https://doi.org/10.1007/s11227-017-2112-9
23. Hummaida, A.R., Paton, N.W., Sakellariou, R.: Adaptation in cloud resource configuration: a survey. J. Cloud Comput. **5**(1), 1–16 (2016)
24. Hummaida, A.R., Paton, N.W., Sakellariou, R.: Shdf - a Scalable Hierarchical Distributed Framework for Data Centre Management. In: 2017 16Th International Symposium on Parallel and Distributed Computing (ISPDC), pp. 102–111. 16Th International Symposium on Parallel and Distributed Computing (2017). https://doi.org/10.1109/ISPDC.2017.15 (2017)
25. Jamshidi, P., Sharifloo, A.M., Pahl, C., Metzger, A., Estrada, G.: Self-Learning Cloud Controllers: Fuzzy Q-Learning for Knowledge Evolution. In: 2015 International Conference on Cloud and Autonomic Computing, pp. 208–211. https://doi.org/10.1109/ICCAC.2015.35 (2015)
26. Jangiti, S., Sriram, V.S.: Scalable and direct vector bin-packing heuristic based on residual resource ratios for virtual machine placement in cloud data centers. Comput. Electr. Eng. **68**, 44–61 (2018). https://doi.org/10.1016/j.compeleceng.2018.03.029
27. Jauro, F., Chiroma, H., Gital, A.Y., Almutairi, M., Abdulhamid, S.M., Abawajy, J.H.: Deep learning architectures in emerging cloud computing architectures: Recent development, challenges and next research trend. Appl. Soft Comput. **106582**, 96 (2020). https://doi.org/10.1016/j.asoc.2020.106582
28. Jin, Y., Bouzid, M., Kostadinov, D., Aghasaryan, A.: Resource management of cloud-enabled systems using model-free reinforcement learning. Ann. Telecommun. **74**(9), 625–636 (2019). https://doi.org/10.1007/s12243-019-00720-y
29. John, I., Sreekantan, A., Bhatnagar, S.: Efficient adaptive resource provisioning for cloud applications using reinforcement learning. In: 2019 IEEE 4Th International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 271–272. https://doi.org/10.1109/FAS-W.2019.00077 (2019)
30. Kardani-Moghaddam, S., Buyya, R., Ramamohanarao, K.: Adrl: a hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds. IEEE Trans Parallel Distrib Syst **32**(3), 514–526 (2021). https://doi.org/10.1109/TPDS.2020.3025914
31. Keller, G., Tighe, M., Lutfiyya, H., Bauer, M.: A Hierarchical, Topology-Aware Approach to Dynamic Data Centre Management. In: Network Operations and Management Symposium (NOMS), pp. 1 –7. Network Operations and Management Symposium (2014) (2014)
32. Khan, T., Tian, W., Buyya, R.: Machine learning (ml)-centric resource management in cloud computing: A review and future directions (2021)
33. Kim, S., Choi, Y.R.: Constraint-aware vm placement in heterogeneous computing clusters. Clust. Comput. **23**(1), 71–85 (2020). https://doi.org/10.1007/s10586-019-02966-6
34. Lebre, A., Pastor, J., Simonet, A., Südholt, M.: Putting the next 500 vm placement algorithms to the acid test: The infrastructure provider viewpoint. IEEE Trans. Parallel Distrib. Syst. **30**(1), 204–217 (2019). https://doi.org/10.1109/TPDS.2018.2855158
35. Liu, N., Li, Z., Xu, J., Xu, Z., Lin, S., Qiu, Q., Tang, J., Wang, Y.: A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning. In: 2017 IEEE 37Th International Conference on Distributed Computing Systems (ICDCS), pp. 372–382. https://doi.org/10.1109/ICDCS.2017.123 (2017)
36. Lolos, K., Konstantinou, I., Kantere, V., Koziris, N.: Elastic Management of Cloud Applications Using Adaptive Reinforcement Learning. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 203–212. https://doi.org/10.1109/BigData.2017.8257928 (2017)
37. Masdari, M., Zangakani, M.: Green cloud computing using proactive virtual machine placement: Challenges and issues. J. Grid Comput. **18**(4), 727–759 (2020). https://doi.org/10.1007/s10723-019-09489-9
38. Matignon, L., Laurent, G.J., Fort-piat, N.L.: Improving Reinforcement Learning Speed for Robot Control. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3172–3177. https://doi.org/10.1109/IROS.2006.282341 (2006)
39. Maurer, M., Brandic, I., Sakellariou, R.: Adaptive resource configuration for cloud infrastructure management. Futur. Gener. Comput. Syst. **29**(2), 472–487 (2013)

40. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
41. Moreno-Vozmediano, R., Montero, R.S., Huedo, E., Llorente, I.M.: Efficient resource provisioning for elastic cloud services based on machine learning techniques. J. Cloud Comput. **8**(1), 5 (2019). https://doi.org/10.1186/s13677-019-0128-9
42. Muller-Merbach, H.: Heuristics and their design: a survey. Eur. J. Oper. Res. **8**(1), 1–23 (1981). https://ideas.repec.org/a/eee/ejores/v8y1981i1p1-23.html
43. Nouri, S.M.R., Li, H., Venugopal, S., Guo, W., He, M., Tian, W.: Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. Fut. Gener. Comput. Syst. **94**, 765–780 (2019). https://doi.org/10.1016/j.future.2018.11.049
44. Pantazoglou, M., Tzortzakis, G., Delis, A.: Decentralized and energy-efficient workload management in enterprise clouds. IEEE Trans. Cloud Comput. **4**(2), 196–209 (2016)
45. Pietri, I., Sakellariou, R.: Mapping virtual machines onto physical machines in cloud computing: A survey. ACM Comput. Surv **49**(3). https://doi.org/10.1145/2983575 (2016)
46. Rao, J., Bu, X., Xu, C.Z., Wang, K.: A Distributed Self-Learning Approach for Elastic Provisioning of Virtualized Cloud Resources. In: 2011 IEEE 19Th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 45–54. https://doi.org/10.1109/MASCOTS.2011.47 (2011)
47. Ren, H., Wang, Y., Xu, C., Chen, X.: Smig-rl: An evolutionary migration framework for cloud services based on deep reinforcement learning. ACM Trans. Internet Technol **20**(4). https://doi.org/10.1145/3414840 (2020)
48. Sedaghat, M., Hernández-Rodriguez, F., Elmroth, E., Girdzijauskas, S.: Divide the Task, Multiply the Outcome: Cooperative Vm Consolidation. In: IEEE International Conference on Cloud Computing Technology and Science, pp. 300–305. IEEE International Conference on Cloud Computing Technology and Science, IEEE, Washington (2014)
49. Shaw, R., Howley, E., Barrett, E.: Applying reinforcement learning towards automating energy efficient virtual machine consolidation in cloud data centers. Inf. Syst., 101722. https://doi.org/10.1016/j.is.2021.101722 (2021)
50. Silva Filho, M.C., Monteiro, C.C., Inácio, P.R., Freire, M.M.: Approaches for optimizing virtual machine placement and migration in cloud environments: a survey. J. Parallel Distrib. Comput. **111**, 222–250 (2018). https://doi.org/10.1016/j.jpdc.2017.08.010
51. Sina, M., Dehghan, M., Rahmani, A.M.: Car-plive: Cloud-assisted reinforcement learning based p2p live video streaming: a hybrid approach. Multimed. Tools Appl. **78**(23), 34095–34127 (2019). https://doi.org/10.1007/s11042-019-08102-1
52. Sniezynski, B., Nawrocki, P., Wilk, M., Jarzab, M., Zielinski, K.: Vm reservation plan adaptation using machine learning in cloud computing. J. Grid Comput. **17**(4), 797–812 (2019). https://doi.org/10.1007/s10723-019-09487-x
53. Song, B., Hassan, M., Huh, E.N.: A Novel Heuristic-Based Task Selection and Allocation Framework in Dynamic Collaborative Cloud Service Platform. In: 2010 IEEE Second International Conference on Cloud Computing Technology and Science, pp. 360–367. https://doi.org/10.1109/CloudCom.2010.53 (2010)
54. Sutton, R.S., Barto, A.G.: Reinforcement Learning: an Introduction, vol. 1. MIT press, Cambridge (1998)
55. Thanh Binh, H.T., Phi Le, N., Minh, N.B., Thu Hai, T., Minh, N.Q., Bao Son, D.: A Reinforcement Learning Algorithm for Resource Provisioning in Mobile Edge Computing Network. In: 2020 International Joint Conference on Neural Networks (IJCNN), pp. 1–7. https://doi.org/10.1109/IJCNN48605.2020.9206947 (2020)
56. Tighe, M., Keller, G., Bauer, M.: Lutfiyya: a distributed approach to dynamic vm management. In: Proceedings of the 9th International Conference on Network and Service Management, pp. 166 to 170. Proceedings of the 9th International Conference on Network and Service Management (2013)
57. Tighe, M., Keller, G., Bauer, M., Lutfiyya, H.: Dcsim: a Data Centre Simulation Tool for Evaluating Dynamic Virtualized Resource Management. In: Network and Service Management (Cnsm), 2012 8Th International Conference and 2012 Workshop on Systems Virtualiztion Management (Svm), pp. 385–392. Network and Service Management (Cnsm), 2012 8Th International Conference and 2012 Workshop on Systems Virtualiztion Management (Svm) (2012)
58. Walsh, W., Tesauro, G., Kephart, J., Das, R.: Utility Functions in Autonomic Systems. In: 2004. Proceedings. International Conference on Autonomic Computing, pp. 70–77. https://doi.org/10.1109/ICAC.2004.1301349 (2004)
59. Watkins, C.J.C.H.: Learning from Delayed Rewards. In: Ph.D. Thesis (1989)
60. Witanto, J.N., Lim, H., Atiquzzaman, M.: Adaptive selection of dynamic vm consolidation algorithm using neural network for cloud resource management. Futur. Gener. Comput. Syst. **87**, 35–42 (2018). https://doi.org/10.1016/j.future.2018.04.075
61. Wu, Y., Tang, M., Fraser, W.: A Simulated Annealing Algorithm for Energy Efficient Virtual Machine Placement. In: 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 1245–1250. https://doi.org/10.1109/ICSMC.2012.6377903 (2012)
62. Wuhib, F., Stadler, R., Spreitzer, M.: Dynamic resource allocation with management objectives: implementation for an openstack cloud. IEEE Trans. Netw. Serv. Manag. **9**(2), 213–225 (2012)
63. Xu, H., Liu, Y., Wei, W., Xue, Y.: Migration cost and energy-aware virtual machine consolidation under cloud environments considering remaining runtime. Int. J. Parallel Prog. **47**(3), 481–501 (2019). https://doi.org/10.1007/s10766-018-00622-x
64. Yadav, M.P., Rohit Yadav, D.K.: Resource provisioning through machine learning in cloud services. Arabian Journal for Science and Engineering.

https://doi.org/10.1007/s13369-021-05864-5 (2021)

65. Yadav, R., Zhang, W., Li, K., Liu, C., Shafiq, M., Karn, N.K.: An adaptive heuristic for managing energy consumption and overloaded hosts in a cloud data center. Wirel. Netw **26**(3), 1905–1919 (2020). https://doi.org/10.1007/s11276-018-1874-1
66. Ying, C., Li, B., Ke, X., Guo, L.: Raven: Scheduling virtual machine migration during datacenter upgrades with reinforcement learning. Mobile Networks and Applications. https://doi.org/10.1007/s11036-020-01632-1 (2020)
67. Zolfaghari, R., Sahafi, A., Rahmani, A.M., Rezaei, R.: Application of virtual machine consolidation in cloud computing systems. Sustain. Comput. Inf. Syst. **30**, 100524 (2021). https://doi.org/10.1016/j.suscom.2021.100524