International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Optimizing Resource Allocation in Cloud Systems using Reinforcement Learning Driven Dynamic VM Placement

Utpal Chandra De *[1], Rabinarayan Satapathy [2], Sudhansu Shekhar Patra[3]

Submitted: 12/03/2024 **Revised**: 27/04/2024 **Accepted**: 04/05/2024

Abstract: Virtual machines (VMs) are extensively used these days as a substitute for physical machines. When the computation power requirement goes beyond that of the existing physical systems, based on client-specific memory requirements, their tools subscriptions, and services the appropriate number of VMs needs to be allocated dynamically. The aim is to minimize the resource cost and energy consumption for optimal usage and enhancement of savings. This is hence an optimization problem that needs to be addressed based on various parameters linked to the system. In this paper, we have worked towards the allocation or placement of VMs in a cloud system, where based on previous requirements we train a model by reinforcement using the A3C algorithm, considering the replays of experiences in various states of the environment to ensure optimal allocation of VMs and hence the real-time functionality of the cloud system.

Keywords: *Dynamic VM Allocation, Reinforcement Learning, Experience Replay, Optimization, A3C algorithm*

1. Introduction

Limitations of physical systems based on their computational power and memory availability are not a hidden thing these days. These issues are addressed using VMs through VDIs to acquire more resources based on requirements and perform the necessary tasks at hand. Also, at times when there are specific client-specific tools or subscribed services, we need to have a VM that will ensure delivery of the tasks regardless of the system being worked upon.

Now based on the actual requirement, a certain number of systems need to be allocated to solve a particular task. This may change with respect to time which in turn makes their allocation a dynamic task during real-time operations. For instance, for a specific task, n number of machines can be allotted but during reduced load conditions, the memory allotted for the machines not in use may go in vain. This leads to unnecessary cost addition and also energy wastage. The other way also holds true where extra memory might be needed suddenly and the only way to allocate it is through manual monitoring and control.

The organization of the paper is like this, after introducing the base concept in Section 1, we have performed a literature survey of similar works previously done in Section 2. The core concepts of reinforcement learning have been discussed in Section 3. The concepts of Q-learning and Deep Q-learning have been explained in Sections 4 and 5, Followed by which A3C algorithm and its uses have been explained in Section 6. Section 7 explains the dataset in hand, the training process followed by the results & discussion of the implementation. The work has been finally concluded in Section 8, where we have also discussed the possible future implications of it. The references cited in the literature survey have been sequentially mentioned after the Appendix.

2. Literature Survey

Previously authors have worked in this domain using static approaches, decision-based systems, optimization algorithms, etc. Authors in [1] have proposed a two-step cascaded model to optimize the dynamic resource allocation process using predictive analytics to predict the number of VMs required, followed by an ML-driven placement technique for Virtual Machine Allocation. Teaching-Based Learning Optimization (TBLO), a population-based meta-heuristic has been used for cost optimization by the authors in [2] for VM Allocation. In [3] the authors have used a context-aware multi-objective genetic algorithm called AGAFF – Aware Genetic Algorithm First Fit to place VMs in a multi-data center cloud environment. The model proposed by authors in [4] improves the decision-making process in VM placement based on resource wastage, placement time, and power consumption using a fitness function that leverages three techniques – Particle Swarm Optimization with Levy Flight (PSOLF), Flower Pollination Optimization (FPO), and a hybrid of both (HPSOLF-FPO). Authors in [5] have surveyed VM Placement techniques and approaches based on network methodologies, energy and power algorithms, cost-optimization, multi-objective optimization, etc. in cloud environments.

A detailed review of preference representations has been explained in [6] explaining their existing usage and adopted

[1] *School of Computer Applications, KIIT (Deemed to be University), India. ORCID ID : 0000-0002-1487-066X*
[2] *Faculty of Emerging Technologies, Sri Sri University, India ORCID ID :*
[3] *School of Computer Applications, KIIT (Deemed to be University), India. ORCID ID : 0000-0001-9996-7681*
** Corresponding Author Email: deutpal@gmail.com*

solving approaches. The authors have also discussed the key challenges and research opportunities in VM Allocation. Authors in [7] have proposed a star topology-based model for initial VM fault tolerance placement for cloud data centers, based on power consumption rate, failure rate, fault tolerance, etc. The authors in [8] have proposed a Multi-Objective Workflow Scheduling (MOWS) scheme addressing the issues of information leaks caused due to data alteration or intermediate data security in the environment. A comprehensive survey of challenges faced during the management of virtualized resources has been performed by the authors in [9]. This includes existing proposals and problem formulation along with the pros and cons of these reviewed papers. Authors in [10] have proposed a way for redundant VM placement optimization based on three algorithms. First, for selecting the appropriate set of servers for hosting the VM based on network topology. The second one finds the optimal strategy to allocate the primary and backup machines. The third algorithm is a heuristic that performs the task-to-VM reassignment optimization.

In [11], the authors have determined the placement of machines in data centers using a multi-objective PSO algorithm. Authors in [12] have proposed an SDN Assistant Routing scheme with multiple servers, finite buffers & multiple vacations for the impatient behavior of clients where certain client retention policies have been used to retain impatient clients in a cloud system. The same authors in [13] have worked towards electricity consumption using an energy-efficient traffic management approach with two server modes – sleep and on, analyzed using a 2D Markov Chain. Authors in [14] have described an offloading system using M/M/c/K queueing model which predicts the number of VMs to be activated while in Fog computing to actively process the jobs. The queue time has also been reduced for the delay-sensitive tasks by performing the optimization through this system. In [15], the authors have also proposed an SDN-based offloading technique using the golden jackal algorithm that has been used to schedule tasks to VMs in the fog layer. The VM allocation problem has been formulated using SAW/WSW to allocate VMs asymmetrically based on CPU utilization and memory usage [16]. The proposed ServerCons algorithm minimizes the number of live migrations, nodes used, and hence the energy usage. The authors have proposed a VM schedule management system for smart grid applications based on panel size [17], to reduce the consumption of brown energy up to an optimized level.

Authors in [18] have performed various experiments to explore the domain of deep reinforcement learning, where they have tried to answer various questions like acceleration of the learning processes, wrong learning by agents, etc. In [19], the authors have investigated Double Deep Reinforcement Learning where the trainer learns in parallel to the agent to eventually create an automatic learning plan for the agent. Here three reward functions – Friendly, Adversarial, and Dynamic have been compared based on the learner's reward. The authors have proposed a segmented and recursive RL algorithm in [20] that reduces the training period and the latency of the systems, hence meeting the time and resource requirements of the AI systems. [21] shows us a framework guiding us regarding associated agent design decisions. Inspired by information theory, this provides us with insights regarding what information to seek, how to seek that & what things to retain from that.

Authors in [22] have presented a survey and provided a broad conceptual overview of model-based RL, which is an integration of RL and planning. This consists of two steps namely dynamics model learning and planning learning integration. In the context of control engineering, [23] uses the principles of RL to design feedback policies for continuous time-dynamical systems hence combining the features of both adaptive and optimal control. Techniques such as Q-learning, Game-Theoretic Learning & intermittent RL have been described in detail, finally describing the details of application in autonomous vehicles. Q-learning, Sampled Data Q-learning followed by System Dynamics Approximation has been explained in [24]. The Actor-Critic Identifier Architecture for the Hamilton-Jacobi-Bellman approximation has been discussed in [25] along with the actor-critic and identifier designs followed by the convergence and stability analysis and simulation results. These book chapters are a part of 'Reinforcement Learning and Approximate Dynamic Programming for Feedback Control'. Authors in [26] have used the techniques of deep RL to create a target-based detection system for unmanned aerial vehicles, developing an integrated cam-based system to automate the landing of GPS-equipped quadcopters. The same authors in [27] have used the same concept for surveillance to capture triggered activities.

State-of-the-art RL techniques are used by major firms today. Deepmind AI has been successfully able to reduce the cooling bill [28] for Google Data centers by 40%. Inspired by this and other similar ideas and techniques, we have tried to optimize the VM allocation process in Cloud-based systems which will be discussed in the next section.

3. Reinforcement Learning

In this paper, we have leveraged the techniques of reinforcement learning to optimally allocate VMs in a dynamic cloud environment. First, we shall discuss the basics of the interaction of an agent with an environment, the reward system, and the whole concept of how we reinforce an agent to learn. Then we move on to the mathematical modeling of the system followed by its integration with deep learning. Finally, we discuss how not just a single agent, but several agents work asynchronously towards our optimization task. This theory is concluded with

the results obtained through our dataset followed by which we finally conclude the work by discussing its future prospects and scopes.

The foundation of Artificial Intelligence [29] is created by training an agent to work and learn in a given environment to achieve its goal. This agent initially performs random actions with zero knowledge but is eventually trained in the environment by imposing a reward system to its actions. The actions lead the agent from one state in the environment to another. Thus, depending upon the state newly acquired, the agent is rewarded based on whether the state is close to the desired state or away from it.

These systems are defined by the elements as shown below:

s – State

e - Environment

a – Action

R – Reward

γ – Discount Factor

Fig. 1 shows the visual representation of a generic AI Training system.

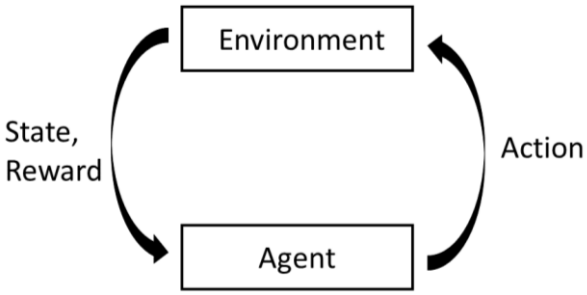


Fig 1. AI Training System

3.1. Bellman Equation

If the agent starts from any general state and we let it explore the environment by taking its time to figure out its path to reach the desired state, it can trace its path back to the initial state by marking all the intermediary states by some flag variable, say V=1. Now if the starting position of the agent is changed, it won't be able to trace its path to the goal state as the flagged values of a lot of states are marked as 1.

Formulated by Richard Ernest Bellman [30], the Bellman equation gives us the direction to train our agent to move towards the desired state starting from any state, hence addressing the problem discussed above. The Bellman Equation is given in Equation 1 below.

$$V(s) = \underbrace{max}_{a} (R(s,a) + \gamma V(s')) \quad (1)$$

Where,

s – Current state

a – Action

s′ – Next State after a

R(s, a) – reward for being in s and performing a

V – flagged value representation of the state.

3.2. Markov Decision Process

There are certain processes where there is a predefined path or set of actions to be taken by the agent to reach the destined state. There is a single takeable action at a given state that can be performed with 100% surety making it a deterministic process. On the other hand, the task in our hand is a non-deterministic process where there is no single takeable action at a given state. Hence there is always a probabilistic value for taking an action at a given state to transition to the other. This has been demonstrated in Figure 2 below.

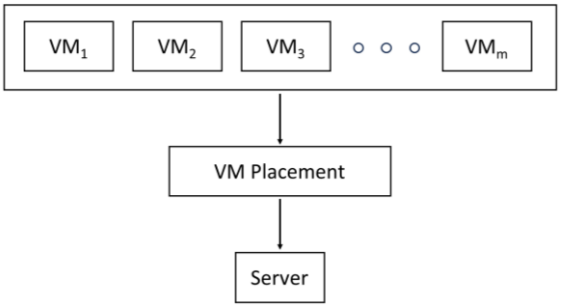


Fig 2(a). Single server system

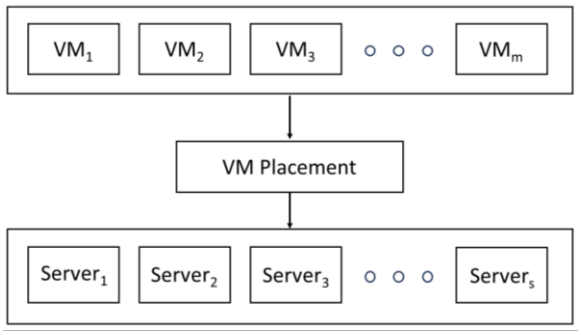


Fig 2(b). Multi server system

Considering Figure 2(a) above consisting of m number of VMs, when a certain VM is chosen for placement, it will obviously be placed in the server that is present. But in the case of Figure 2(b) with s number of servers, unless a rule-based optimization system is installed for the VM Placement, there will be a probabilistic value for the placement of the selected VM corresponding to each server, hence making it a non-deterministic process.

Secondly, in these cases, the requirement pattern can be statistically analyzed but precise prediction becomes difficult unless we reinforce the trend properly. These are stochastic processes and follow Markov's Property. If the conditional probability distribution of future states of the process depends only upon the present state and not on the

sequence of events that preceded it, such a process with this property is called a Markov Process. Hence, we need to use Markov Decision Processes [31] (MDPs) which provide a mathematical framework for modeling the decision-making process in situations where outcomes are partly random and partly under the control of a decision-maker. Following this property, Equation (1) takes a form where the weighted sum of probabilities is used corresponding to each probable upcoming state.

$$V(s) = \underbrace{max}_{a} \left(R(s,a) + \gamma \sum_{s'} P(s,a,s')\, \mathrm{V}(s')\right) \quad (2)$$

Where,

$$\sum_{s'} P(s,a,s') = P_1(\mathrm{Server}_1) + P_2(\mathrm{Server}_2) + P_3(\mathrm{Server}_3) + \cdots + P_s(\mathrm{Server}_s)$$

is the reward for being in s, performing a & moving to s′.

3.3. Policies and Living Penalties

To proceed through a sequence of states, we need a proper plan of execution. But it is important to note that a plan works only when we know exactly what steps to follow next. But in the real-time context, we face randomness and various other non-deterministic hindrances, and this is where a policy (π) comes into play, where the action is decided not based on a predefined set of rules and regulations, but by reinforcing the agent to learn from dynamic environments. Based on the randomness of each state, the probability values may change. We calculate the current state probability values and compare them to the previously calculated values for each state, according to which the agent decides its next action.

Policies are made and imposed for each state by imposing a living penalty in our case. In this way the agent not just finds a way to the desired state but is also bound to optimize its path and reach the desired state quickly. The living penalty is basically a negative reward R(s) in each state. This value of R(s) in itself is a hyperparameter. If the negativity increases above a certain limit, the model tends to overfit. To minimize the penalty that can be incurred, the agent tends to take a path that doesn't actually reach the desired state, but the overall penalty is less compared to taking other takeable paths and reaching the desired state.

4. Q-Learning

So far, we looked at the values of each state V(si). But to take this ahead from the perspective of the agent, we need to consider the value (or quality) of various actions that can be performed by it in a given state. We denote this quality as Q(s, a). Based on the action that the agent takes, it can end up at multiple states, and then the new values are calculated, and so on. This takes the form of a recursive event that we execute through a recursive function of V. But for that, we need to map our V(s) to Q(s, a) from Equation (2).

For performing an action in a state, the agent certainly gets to have a reward regardless of whether it is negative, zero, or positive. Hence R(s, a) is a quantified metric that certainly adds up to the quality of action taken by the agent. The next thing that happens after the agent is rewarded is that the agent ends up in another state. There can be multiple such states and different probability values corresponding to them, but wherever the agent ends up there's going to be a V value of that state V(s′). But since there are multiple such probable states, we will add the weighted probabilities corresponding to those states as we did earlier in Equation (2). Hence our quality is expressed mathematically [32] as shown in the equation below.

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s')\, \mathrm{V}(s') \quad (3)$$

From Equation (3) above, we see that the Q value is exactly similar to the value inside the max() in Equation (2) Or in simpler terms Equation (2) can also be written as:

$$V(s) = \underbrace{max}_{a} \left(Q(s,a)\right) \quad (4)$$

If we look at it, we see that in Equation (2) or Equation (4), we are looking at the maximum of the result we get across all possible actions, by taking each of those actions, and in Equation (3) we are looking at what would we get by taking a certain action, and hence the quality of a specific action.

To entirely get rid of the recursive V(), we put Equation (4) in Equation (3) in terms of the new state or s′, as shown in the equation below.

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \underbrace{max}_{a'} Q(s',a') \quad (5)$$

Equation (5) above, gives us the recursive formula in terms of the Q-value of an action.

4.1. Temporal Difference

In the case of a deterministic environment, it's straightforward to calculate the V-values as a specific plan can be followed. But in cases where there is stochasticity or randomness in the environment, the computation of these values is not easy. Temporal differences simplify the task for the agent to compute these values. Temporal difference is the difference between quality between two subsequent states – s′ and s.

Since we are looking at the Q-values, we shall move ahead with Equation (5) . For the sake of simplicity, we write it in the format of the deterministic Bellman Equation similar to Equation (1), where we replace the whole summation term with simply the max() function corresponding to the new state. This notation has been used in various literatures and

we shall also proceed ahead the same way. Hence, Equation (5) takes the form as mentioned below:

$$Q(s,a) = R(s,a) + \gamma \underbrace{max}_{a'} (Q(s',a')) \quad (6)$$

So earlier for a given state, the Q-value stored in the memory is denoted by Q(s, a). The same value after recalculations for the new state that we end up in, is given by Equation (6). Hence the temporal difference is the difference between these two mentioned in the equation below.

$$TD(a,s) = R(s,a) + \gamma \underbrace{max}_{a'} \left(Q(s',a')\right) - Q(s,a) \quad (7)$$

We call it the temporal difference [33] because what we calculate over here is essentially the same thing but at different intervals of time or states. We can use this difference to our advantage if there's a shift in time. One thing that we did earlier in Eq. 6 is to simply calculate the new Q-value by getting rid of the old one. But that's not a smart act as our environment is subject to randomness. There might be cases where the old Q-values consistently happen say 90-95% of the time then all of a sudden, a random event is triggered causing the new event. In this way what we would do is simply discard the value that's responsible for the bulk of the situation and move ahead with a value that happens only 5-10% of the time. This is why we don't completely change our Q-values but rather change them step by step, bit by bit.

We would express this in terms of timestamps where Equation (7) takes the form as mentioned below.

$$TD(a,s) = R(s,a) + \gamma \underbrace{max}_{a'} \left(Q(s',a')\right) - Q_{t-1}(s,a) \quad (8)$$

Where,

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD_t(a,s) \quad (9)$$

Putting Eq. 8 in Eq. 9, we get

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha \left(R(s,a) + \gamma \underbrace{max}_{a'} \left(Q(s',a')\right) - Q_{t-1}(s,a) \right) \quad (10)$$

From Eq. 10 above, we can say that,

If α = 0,

$$Q_t(s,a) = Q_{t-1}(s,a) \quad (11)$$

If α = 1,

$$Q_t(s,a) = R(s,a) + \gamma \underbrace{max}_{a'} \left(Q(s',a')\right) \quad (12)$$

From Equation (11), we see that when α = 0, see that there's no difference between the Q-values of the two subsequent states and the agent doesn't learn anything. From Equation (12), we see that when α = 1, we acquire back Equation (6) again, which is not the best approach as we discussed above.

Hence, we see that the α value is a hyperparameter where 0 < α < 1. After a certain number of iterations, when Equation (8) converges, we see that there's no significant temporal difference between the Q-values of the subsequent states and with α ≠ 0 we get,

$$Q_t(s,a) \approx Q_{t-1}(s,a) \quad (13)$$

This means that the agent has sufficiently learnt, and we don't need to run iterations to update the values anymore. However, in cases where our environments change with respect to time, (not just random and stochastic changes, but the whole environment changes) we need to keep these iterations running & update our Q-values so that we reinforce the agent to be adaptive to the changing environment and hence the optimal policy as well coz with changing environment the optimal policy also changes.

5. Deep Q-Learning

The concepts related to Q-learning have been explained in detail in the previous sections. In this section, we integrate deep learning with Q-learning and leverage the power of neural networks to predict the best takeable action to move from one state to the next.

Referring to Figure 2(b), we see that there are m number of machines that can be allocated to s number of servers at any time stamp t. In addition to this, there might be certain timestamps where no action needs to be taken, i.e., no VM needs to be allocated or deallocated to or from a server. Hence, we need to map the set of inputs and outputs to a neural network and design the architecture of it. For every given set of VM and Server, we need to identify two things –

1. Whether do we need to allocate or deallocate the VM.
2. If we need to allocate, whether the server can take the load of the machine to be allocated.

Hence, our inputs to the architecture at any timestamp t are – VM_i & S_j. The deallocation of a machine can be performed by running a scan time to time across all servers, to identify the idle resources and deallocate them through a trigger. Hence, the prime challenge now is with allocation of these

VMs to our Servers. The input set at a timestamp t – $\{VM_i, S_j\}$ are fed to the network and the output leads to a range of Q values corresponding to the s servers i.e., $\{Q_1, Q_2, \ldots, Q_s\}$. These Q-values are passed through a SoftMax layer to acquire the probability values for each predicted quality corresponding to the s servers and perform the best allocation based on them. The NN architecture has been shown in Fig. 3 with the set of inputs and outputs as mentioned above.

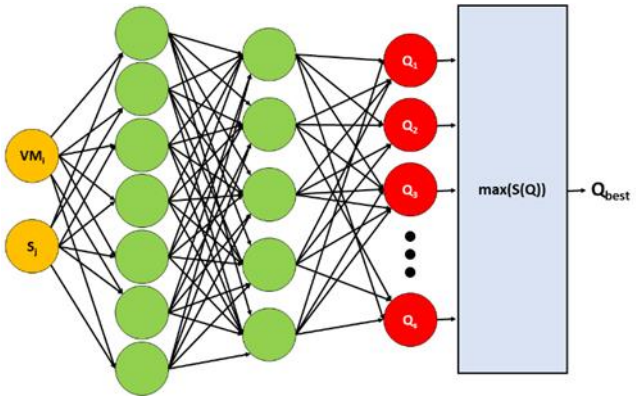


Fig. 3 Deep Q-Learning Architecture

In Fig. 3 above, S(Q) is the SoftMax function defined as follows:

$$S(Q)_k = \frac{e^{Q_k}}{\sum_{k=1}^{s} e^{Q_k}} \quad (14)$$

The output hence acquired from this is the maximum probabilistic values of the Q values, and the allocation of that VM corresponding to the server is performed. This acts as a policy to select the next action as discussed previously in Section-5.2. It's also important to note here that SoftMax is just one of such action selection policies. Other policies may be ϵ-greedy [35], ϵ-soft(1 - ϵ), etc.

Like the training process of neural networks, after forward propagation, the value loss is calculated as per the following equation.

$$L_V = \sum(Q_{Target} - Q)^2 \quad (15)$$

Then the control backpropagates the network to adjust the weights of the network. This whole sequence is called an epoch. This training sequence is repeated up to a certain number of epochs to acquire or reach the maxima.

5.1. Experience Replay

There are two things that happen on a broader level during the whole process – training and acting. During the training phase, the network learns with every new state and updates the weights by which it gets better and better at dealing with the environment.

Now at times due to monotonicity in the environment where in each new state, there's no significant change in the environment and a sequential nature of experiences, a bias might come up during the training phase due to interdependent and correlated consecutive states in the environments. As a solution to this, the neural network isn't trained based on the whole set of experiences but on a batch of such uniformly distributed experiences. A batch can be a rolling window of experiences and can break the interdependence and correlation hence the pattern of the bias. In addition to this, sometimes there might be very valuable and rare experiences that might be difficult for the agent to learn if the experiences are not stored and saved in batches. This is called experience replay [34] where the experiences saved in batches are replayed and used to train the network enabling it to eradicate such biases and learn every aspect of the environment well.

5.2. Action Selection Policies

After analyzing the training aspect of the process in the previous section, we look at the acting part of it. As discussed in Section 7 about the significance of policies, we will look at the various ways to select the appropriate policy to act.

From Fig. 3, we can see that the SoftMax function has been used after the Q values corresponding to the various servers have been calculated. The question is, why don't we directly use the Q-values to perform our next action using the value that is the best? This can be explained through some scenarios. Let's say the best value at a time instant t is Q3, and the action corresponding to that is performed. But eventually, it turns out to be a bad action due to which the agent gets a negative reward. But since the policy says to perform the action corresponding to the best Q-value the agent is forced to explore the environment and learn by itself that Q3 is not a good action to be performed based on the negative rewards to be acquired. In such cases, the agent is forced to learn by itself and update the weights according to the rewards it gets.

On the other hand, there might be cases where the agent gets stuck in a local maximum. Through its initial exploration, the agent might think that it is taking the correct action based on the positive rewards but due to its limited perception it may get biased towards the actions it took so far and get reinforced by them. What if there are other local maximums or a global one that is not yet into the experience of the agent which might lead it to take better actions. This is where the action selection policies come into the picture where we can leverage the good actions discovered so far by the agent but also at the same time help it to not get stuck in a local maximum.

6. A3C Algorithm

Developed at Google Deepmind in 2016, stands for Asynchronous Advantage Actor Critic Algorithm [36]. The architecture used for this algorithm is faster compared to simple Deep Q-learning and takes less training time,

achieving better results. We shall look at each of the individual entities separately and finally club them together in an implementable way to carry our task ahead.

6.1. Actor - Critic

In Fig. 3 we saw that we get s number of outputs corresponding the s policies of VM placement to the s servers. Keeping those outputs intact, we separately take one output for the value of the state. In this way, we have two sets of outputs, one corresponding to the policy – π(s), which is taken care of by the actor network and the other corresponding to the value – V(s), which is taken care of by the critic network. The segregation of the output layer into actor and critic networks after slightly modifying Fig. 3 has been shown in Fig. 4. From here onwards, for simplification we shall use a vector representation as input to the neural network, square boxes to represent a layer of neurons and a single arrow to represent the fully connected layers.

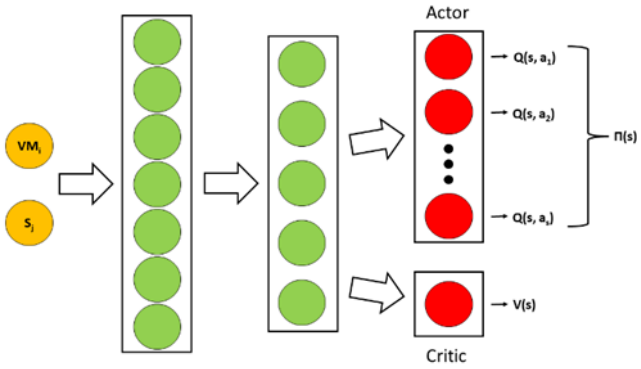


Fig. 4 Actor Critic Architecture

6.2. Asynchronous

To make the actor critic network asynchronous [37], instead of having just a single agent, we deploy a set of agents, initialized differently to attack the environment. This initialization is performed by setting different seeds randomly for each agent. In this way, for n agents we get to have n-times the experience compared to the experience otherwise acquired by a single agent. In this way, we reduce the chance of the agents getting stuck at a local maximum. The likelihood of several agents getting stuck at a local maximum decrease with an increase in the number of agents, as long as agents interact with each other & share their experience. In this way, getting towards the desired state becomes comparatively fast. This experience is shared through the critic – V(s). The actors might be different, but the critic neuron must be the same for all agents.

Initially, each agent used to have its own environment to explore. But the creator of PyTorch adjusted the code available in GitHub where all the networks were reduced to one single network, i.e., all the agents will now have just one environment to learn from or we can say that all the agents will asynchronously attack one environment to find the maximum

6.3. Advantage

Since the output has been segregated into two sets of neurons, we have two different losses over here. One is the value loss as mentioned in Equation (15) in Section 5. The other one is the policy loss for the calculation of which we need to consider the advantage [38] given by

$$A = Q(s,a) - V(s) \quad (16)$$

The critic shares a V-value which is common for all the agents. Hence it knows how much better our selected Q-value is when compared to the known V-value. The policy loss is then similarly backpropagated through the network. The weights are adjusted such that the advantage is maximized. In this way, the critic neuron through the known value – V(s), observes the policies made by the agents, enabling them to cooperate. The practicability of the actions of an agent is determined by its entropy, given by Equation (17) [39]. The representation of the Asynchronous Advantage Actor-Critic Network is shown in Fig. 5.

$$H(x) = -\sum_{i=1}^{n} P(x)\, ln(P(x_i)) \quad (17)$$

The entropy loss for the SoftMax Policy output is calculated and summed over all states. The final adjusted loss is calculated by Eq. 18 mentioned below.

$$L = L_V + L_\pi - \beta L_H \quad (18)$$

Where,

LV is the value loss.

Lπ is the policy loss.

LH is entropy loss.

β [40] is the momentum or influence of entropy loss.

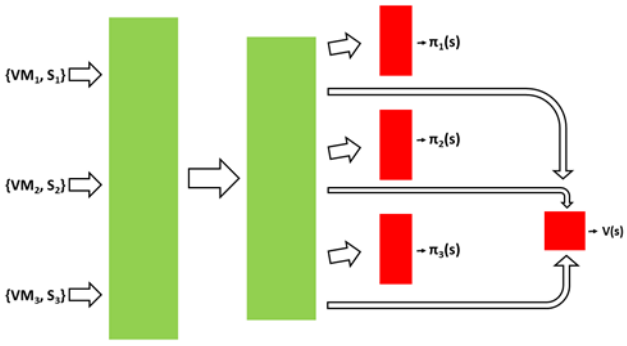


Fig. 5 A3C Architecture

7. Results and Discussion

Most of the codes publicly available have been written in TensorFlow, but for the sake of simplicity, we have used PyTorch for our implementation. For experimentation purposes, we used the Google Cluster Usage Traces dataset [42 – 43] which consists of job events and resource consumption from the data centers in Google, managed by

Borg cluster management software.

7.1. The Dataset

This dataset is a collection of data that describes the workloads running on Google's compute clusters. It gives us information about the clusters such as jobs (submission, scheduling and completion), tasks placement, migration and eviction), machines (attributes, availability and failures) and requests and usage of CPU, memory, network and disk. These things have been used to study the patterns characteristics of workloads, evaluate and improve the policies, model and simulate the behavior and hence the performance of the cluster. The granular aspects have been explained in detail in [44 – 45].

The dataset consists of 405894 data points corresponding to 33 features, which act as an input to our model. The data is labelled, i.e., there is a clear indication corresponding to each datapoint, whether the job, collection or task instance failed or not. Out of these, there are 313216 datapoints where the allocation was performed successfully. All these 33 attributes are not of use to us, hence we group the relevant attributes together under VM and S, to feed out architecture with inputs. The relevant attributes as per their category have been segregated below in Table 1.

Table 1 – VM, S and Policy Mappings

VM	***S***	***Π***
Collection ID	Alloc Collection ID	Enable
Scheduling Class	Resource Request	Evict
Priority	Constraint	Fail
Instance Index	Vertical Scaling	Finish
Machine ID	Assigned Memory	Kill
Start After Collection IDs	Page Cache Memory	Lost
	Cluster	Queue
		Schedule
		Update Pending
		Update Running

These attributes were stacked and used as inputs to the model. The output attribute corresponding to policies (π) is the event attribute, based on which the system decides an action to take out of 10 different actions and move to the next state. The list of actions is also mentioned in the third column of Table 1. The rest of the attributes are intrinsic to the event, whose work is single handedly taken care of by the critic network. Based on the values of these attributes, the agent is trained and set to explore the global optimal path to reach the desired state.

7.2. Model Training

Deploying a single agent in the environment and training is a very tedious task and needs high computational power. This has been demonstrated in Figure 6.

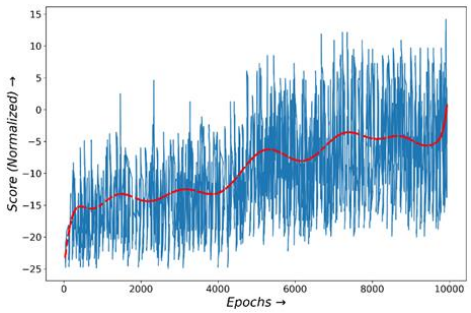


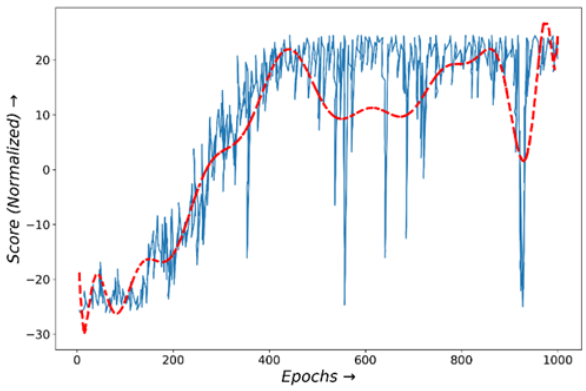
Fig. 6 Single Agent Training

In addition to this, from the trend line in Figure 6 we can see that even after 10k epochs, there is still scope for the agent to learn with a gradual increase in the slope. Hence, instead of wasting time to find the maxima, we leveraged the power of A3C algorithm as explained in Section 6 above. Instead of deploying a single agent, we've deployed three agents to attack the environment simultaneously. In this way, the agents cooperate and hence, learn

faster to find the optimal position. This saved us days of time and we could train the model within a duration of 8½ hours.

We achieved the maximum at somewhere around 950 epochs, followed by which there is a decreasing trend. The training graph for A3C model up to 1000 epochs is shown in Figure 7. This shows that in this way the agents cooperate and learn quickly in the environment.

Fig. 7 Three Agent Training



7.3. 1.2. Model Output

We studied three types of allocations – random, static and our A3C based allocations. Based on these we have compared several metrics through which we can check the efficacy of the system.

Table 2 – Average Resource, CPU and Memory Utilizations

Allocation Strategy	Average Resource Utilization	Average CPU Utilization	Average Memory Utilization
Random Allocation	68.7%	63.5%	72.4%
Static Allocation	65.2%	60.2%	70.8%
A3C Allocation	**85.6%**	**79.8%**	**75.5%**

Table 3 – Cost Savings and Energy Efficiency

Allocation Strategy	Cost Savings	Energy Efficiency
Random Allocation	8.2%	63.5%
Static Allocation	12.6%	60.2%
A3C Allocation	34.8%	**79.8%**

Table 4 – Average Response Time and Throughput across Batch and Interactive Workloads

Workload Type	Allocation Strategy	Average Response Time	Throughput
Batch	Random Allocation	290 ms	210 Tasks/min
	Static Allocation	320 ms	180 Tasks/min
	A^3C Allocation	**190 ms**	**320 Tasks/min**
Interactive	Random Allocation	210 ms	350 Tasks/min
	Static Allocation	180 ms	400 Tasks/min
	A3C Allocation	140 ms	510 Tasks/min

Values in Table 2 show that the A3C allocation outperforms both static and random allocations, & hence is more effective in adapting to workload changes and utilizing allocated resources. The static allocation is unable to accommodate dynamic workload patterns, hence resulting in lower utilization. Random placement while slightly better, still lacks the intelligence provided by A3C decision which leads to suboptimal resource allocation. The values in Table 2 also show that the proposed A^3C approach consistently outperforms the rest of the two approaches justifying the A^3C approach's capability to optimize the allocations of both CPU and memory resources. Static allocation falls short in allocating resources efficiently, resulting in lower utilization. Random allocation, while slightly better again lacks the intelligence of the A^3C approach leading to suboptimal resource allocation, again justifying the resource utilization. Figure 8 shows the Average utilizations of resource, CPU and memory across the three modes of allocations, namely – Static, Random and A^3C.

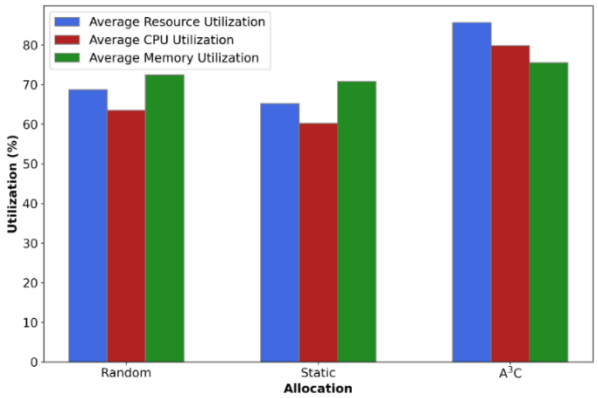


Fig. 8 – Average Utilizations

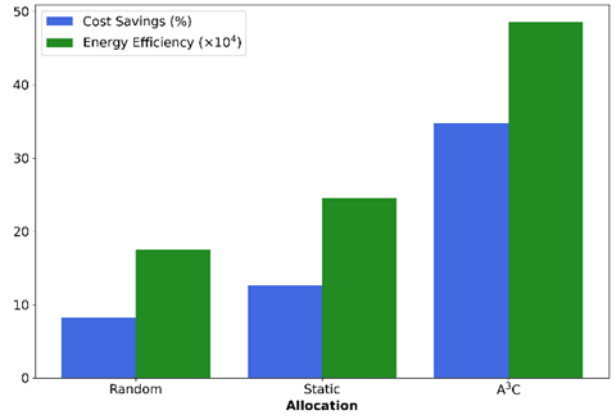


Fig. 9 – Cost and Energy Savings

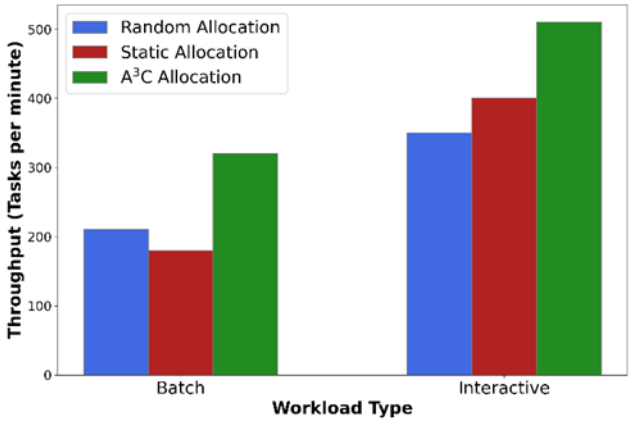


Fig. 10 – Throughput

Table 3 shows the ability to allocate resources efficiently according to workload demands. Random and static allocations exhibit lower cost savings due to their lack of adaptability to dynamic situations. Hence, we see that the proposed A^{3}C approach not only optimizes resource utilization but also minimizes unnecessary expenditure which makes it a cost-effective solution for cloud resource allocation. Table 3 also presents a comprehensive analysis of energy efficiency achieved by different allocation strategies. In correlation with cost savings, the proposed approach significantly contributes to energy efficiency compared to static allocation. This approach's ability to allocate resources dynamically according to workload demands minimizes resource wastage, hence causing both environmental and financial benefits. Random and static allocation exhibit lower energy efficiency due to a lack of adaptability. Figure 9 shows the graphical representation of the cost savings and energy efficiency across the three allocation modes.

From Table 4 above, we see that for both batch and interactive workloads the A^{3}C technique yields improved response time and throughput compared to static and random strategies. This shows its adaptability to various workload characteristics. The lower response times and higher throughputs enhance the efficiency of the A^{3}C model ensuring optimal performance for diverse workloads. Figure 10 graphically represents the throughput across batch and interactive workloads for the three allocation modes being discussed. Figure 11 graphically represents the average response time across batch and interactive workloads for the three allocation modes being discussed.

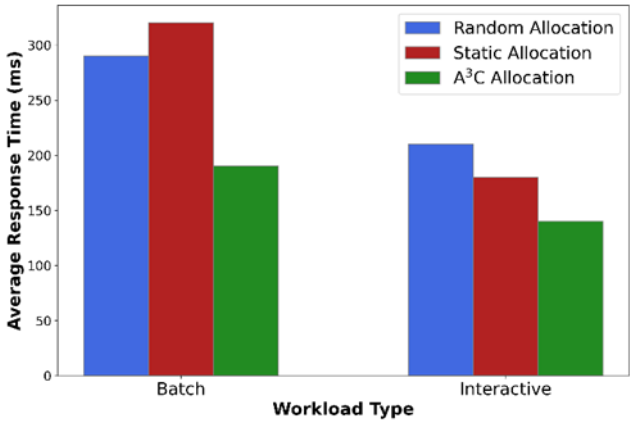


Fig.11 – Average Response Time

8. Conclusion and Future Scope

After looking at the outputs obtained after using the proposed A^{3}C approach in the traces dataset, we acquired satisfactory results compared to the rest allocation strategies in all aspects whether be it in terms of energy saving, cost cutting, utilization, throughput, or response time. Practically it is impossible to meet the ideal conditions, but one always aims to optimize the model as much as possible.

There are various other methods such as the massively distributed General RL Architecture (Gorila) [46], dueling [47] for model-free RL, LSTM-based A^{3}C architectures, etc. which have outperformed other conventional algorithms in gaming environments. These can be tweaked and implemented as per the optimization requirement in hand in the future to see if the values can be optimized further.

Appendix

The publicly downloadable version of the dataset can be

obtained at: https://www.kaggle.com/datasets/derrickmwiti/google-2019-cluster-sample**.**

Author contributions

Conceptualization, RS, SSP; methodology, UCD, RS; formal analysis, SSP; writing—original draft preparation, UCD; writing—review and editing, UCD, RS, and SSP; visualization, SSP; supervision, RS, SSP; project administration, RS. All authors have read and agreed to the published version of the manuscript.

Conflicts of interest

The authors declare no conflict of interest.

References

[1] U. C. De, R. Satapathy & S. S. Patra, "Optimizing Resource Allocation using Proactive Predictive Analytics and ML-Driven Dynamic VM Placement," 4th Global Conference for Advancement in Technology, 2023.

[2] U. C. De, R. Satapathy and S. S. Patra, "Cost Analysis and Optimization of Virtual Machine Allocation in the Cloud Data Center," International Conference on Inventive Computation Technologies (ICICT), pp. 809-813, IEEE, 2023.

[3] S. M. Seyyedsalehi and M. Khansari, "Virtual Machine Placement Optimization for Big Data Applications in Cloud Computing," in IEEE Access, 10, pp. 96112-96127, 2022.

[4] S. Mejahed, M. Elshrkawey, "A multi-objective algorithm for virtual machine placement in cloud environments using a hybrid of particle swarm optimization and flower pollination optimization", PeerJ Computer Science, 8, e834, 2022.

[5] Sudhakar, Saravanan, "A Survey and Future Studies of Virtual Machine Placement Approaches in Cloud Computing Environment", 6th International Conference on Cloud Computing and Internet of Things, pp. 15-21, 2021.

[6] A. Alashaikh, E. Alanazi, A. Al-Fuqaha, "A survey on the use of preferences for virtual machine placement in cloud data centers", ACM Computing Surveys (CSUR), 54(5), pp. 1-39, 2021.

[7] W. Zhang, X. Chen, J. Jiang, "A multi-objective optimization method of initial virtual machine fault-tolerant placement for star topological data centers of cloud systems", Tsinghua Science and Technology, 26(1), pp. 95-111, 2020.

[8] Farzaneh Abazari, Morteza Analoui, Hassan Takabi, Song Fu, "MOWS: Multi-objective workflow scheduling in cloud computing based on heuristic algorithm", Simulation Modelling Practice and Theory, 93, pp. 119-132,2019.

[9] M. C. Silva Filho, C. C. Monteiro, P. R. Inácio, M. M. Freire, "Approaches for optimizing virtual machine placement and migration in cloud environments: A survey", Journal of Parallel and Distributed Computing, 111, pp. 222-250, 2018.

[10] Ao Zhou, Shangguang Wang, Member, Bo Cheng, Member, Zibin Zheng, Member, Fangchun Yang, Senior Member, Rong N. Chang, Senior Member, Michael R. Lyu, Fellow, Rajkumar Buyya, "Cloud Service Reliability Enhancement via Virtual Machine Placement Optimization", IEEE Transactions on Services Computing, 10(6), pp. 902-913, 2017.

[11] J. Gao, G. Tang, "Virtual Machine Placement Strategy Research", International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2013, pp. 294-297,2013.

[12] B. B. Dash, R. Satapathy, S. S. Patra, "SDN-Assisted Routing Scheme in Cloud Data Center using Queueing Vacation Policy", 2nd International Conference on Edge Computing and Applications (ICECAA), pp. 1-6, 2023.

[13] B. B. Dash, R. Satapathy and S. S. Patra, "Energy Efficient SDN-assisted Routing Scheme in Cloud Data Center", 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN), pp. 1-5, 2023.

[14] S. Behera, N. Panda, U. C. De, B. B. Dash, B. Dash, S. S. Patra, "A task offloading scheme with Queue Dependent VM in fog Center", 6th International Conference on Information Systems and Computer Networks (ISCON), pp. 1-5, 2023.

[15] B. B. Dash, S. S. Patra, R. Satpathy and B. Dash, "Improvement of SDN-based Task Offloading using Golden Jackal Optimization in Fog Center," World Conference on Communication & Computing (WCONF), pp. 1-6,2023.

[16] S. S. Patra, R. Govindaraj, S. Chowdhury, M. A. Shah, R. Patro and S. Rout, "Energy Efficient End Device Aware Solution Through SDN in Edge-Cloud Platform," in IEEE Access, vol. 10, pp. 115192-115204, 2022.

[17] Inès De Courchelle, Tom Guérout, Georges Da Costa, Thierry Monteil, Yann Labit, "Green energy efficient scheduling management", Simulation Modelling Practice and Theory, Volume 93, pp. 208-232, 2019.

[18] M. Kaloev, G. Krastev, "Experiments Focused on Exploration in Deep Reinforcement Learning", 5th International Symposium on Multidisciplinary Studies

and Innovative Technologies (ISMSIT), pp. 351-355, 2021.

[19] J. Kiefer and K. Dorer, "Double Deep Reinforcement Learning", 2023 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 17-22, 2023.

[20] M. H. Krishna and M. M. Latha, "Complexity and Performance Evaluation of Segmented and Recursive Reinforcement Learning", IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON), pp. 1-7, 2021.

[21] Xiuyuan Lu, Benjamin Van Roy, Vikranth Dwaracherla, Morteza Ibrahimi, Ian Osband, Zheng Wen, "Reinforcement Learning Bit by Bit", now, 2023.

[22] Thomas M. Moerland, Joost Broekens, Aske Plaat and Catholijn M. Jonker, "Model-based Reinforcement Learning: A Survey", Foundations and Trends® in Machine Learning, 16(1), pp 1-118, 2023.

[23] Kyriakos G. Vamvoudakis and Nick-Marios T. Kokolakis, "Synchronous Reinforcement Learning-Based Control for Cognitive Autonomy", Foundations and Trends® in Systems and Control, 8(1–2), pp 1-175, 2020.

[24] Frank L. Lewis, Derong Liu, "Reinforcement Learning Control with Time-Dependent Agent Dynamics," in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, IEEE, pp.203-220, 2013.

[25] Frank L. Lewis, Derong Liu, "An Actor-Critic-Identifier Architecture for Adaptive Approximate Optimal Control", in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, IEEE, pp.258-280, 2013.

[26] T. Swain, M. Rath, J. Mishra, S. Banerjee and T. Samant, "Deep Reinforcement Learning based Target Detection for Unmanned Aerial Vehicle", IEEE India Council International Subsections Conference (INDISCON), pp. 1-5, 2022.

[27] S. Banerjee, T. Swain, J. Mishra, M. K. Rath and T. Samant, "Surveillance using Unmanned Aerial Vehicle for Triggered Activity Capturing," 1st IEEE International Conference on Industrial Electronics: Developments & Applications (ICIDeA), pp. 6-11, 2022.

[28] J. Luo, C. Paduraru, O. Voicu, Y. Chervonyi, S. Munns, J. Li, ... & D. J. Mankowitz, "Controlling commercial cooling systems using reinforcement learning." arXiv preprint arXiv:2211.07357, 2022.

[29] R. S. Sutton, A. G. Barto, "Reinforcement learning: An introduction", MIT press, 2018.

[30] R. Bellman, "The theory of dynamic programming", Bulletin of the American Mathematical Society, 60(6), pp. 503-515, 1954.

[31] D. J. White, "A survey of applications of Markov decision processes", Journal of the operational research society, 44(11), pp. 1073-1096, 1993.

[32] Martijn Van Otterlo, "Markov decision processes: Concepts and algorithms." Course on Learning and Reasoning, 2009.

[33] Richard S. Sutton, "Learning to predict by the methods of temporal differences", Machine learning, 3(9-44),1988.

[34] T. Schaul, J. Quan, I. Antonoglou, D. Silver, "Prioritized experience replay", arXiv preprint arXiv:1511.05952, 2015.

[35] M. Tokic, "Adaptive ε-greedy exploration in reinforcement learning based on value differences", In Annual Conference on Artificial Intelligence, pp. 203-210, 2010.

[36] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, & J. Kautz, "Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU", ArXiv. /abs/1611.06256, 2016.

[37] J. Schulman, P. Moritz, S. Levine, M. Jordan, & P. Abbeel, "High-dimensional continuous control using generalized advantage estimation", arXiv preprint arXiv:1506.02438, 2015.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, & M. Riedmiller, "Playing Atari with Deep Reinforcement Learning", ArXiv. /abs/1312.5602, 2013.

[39] R. J. Williams, & J. Peng, "Function optimization using connectionist reinforcement learning algorithms", Connection Science, 3(3), pp. 241-268, 1991.

[40] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, "Large-scale cluster management at Google with Borg", In Proceedings of the tenth european conference on computer systems, pp. 1-17, 2015.

[41] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, J. Wilkes, "Borg: the next generation", In Proceedings of the fifteenth European conference on computer systems, pp. 1-14, 2020.

[42] Victor Chudnovsky, Rasekh Rifaat, Joseph Hellerstein, Bikash Sharma, Chita Das, “Modeling and Synthesizing Task Placement Constraints in Google

Compute Clusters”, Symposium on Cloud Computing (SoCC), 2011.

[43] M. Carvalho,W. Cirne, F. Brasileiro, J. Wilkes, "Long-term SLOs for reclaimed cloud computing resources", in Proceedings of the ACM Symposium on Cloud Computing, pp. 1-13, 2014.

[44] A. Nair, P. Srinivasan, S.Blackwell, C. Alcicek, R. Fearon, , A. De Maria, ..., D. Silver, "Massively parallel methods for deep reinforcement learning", arXiv preprint arXiv:1507.04296, 2015.

[45] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas, "Dueling network architectures for deep reinforcement learning", in International conference on machine learning, pp. 1995-2003, 2016.