



A Deep Reinforcement Learning Approach for Cost Optimized Workflow Scheduling in Cloud Computing Environments

Amanda Jayanetti, Saman Halgamuge*, and Rajkumar Buyya
Cloud Computing and Distributed Systems (CLOUDS) Lab,
School of Computing and Information Systems,
The University of Melbourne, Australia
*School of Electrical Mechanical and Infrastructure Engineering,
The University of Melbourne
Australia

ABSTRACT

Cost optimization is a common goal of workflow schedulers operating in cloud computing environments. The use of spot instances is a potential means of achieving this goal, as they are offered by cloud providers at discounted prices compared to their on-demand counterparts in exchange for reduced reliability. This is due to the fact that spot instances are subjected to interruptions when spare computing capacity used for provisioning them is needed back owing to demand variations. Also, the prices of spot instances are not fixed as pricing is dependent on long term supply and demand. The possibility of interruptions and pricing variations associated with spot instances adds a layer of uncertainty to the general problem of workflow scheduling across cloud computing environments. These challenges need to be efficiently addressed for enjoying the cost savings achievable with the use of spot instances without compromising the underlying business requirements. To this end, in this paper we use Deep Reinforcement Learning for developing an autonomous agent capable of scheduling workflows in a cost efficient manner by using an intelligent mix of spot and on-demand instances. The proposed solution is implemented in the open source container native Argo workflow engine that is widely used for executing industrial workflows. The results of the experiments demonstrate that the proposed scheduling method is capable of outperforming the current benchmarks.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Deep Reinforcement Learning, Workflow Scheduling, Cost Optimization, Spot market resources

ACM Reference Format:

Amanda Jayanetti, Saman Halgamuge*, and Rajkumar Buyya. 2024. A Deep Reinforcement Learning Approach for Cost Optimized Workflow Scheduling in Cloud Computing Environments. In *2024 Asia Pacific Conference on Computing Technologies, Communications and Networking (CTCNet 2024)*, July 26–27, 2024, Chengdu, China. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3685767.3685780>

1 INTRODUCTION

Cloud computing leverages virtualization techniques for providing users with convenient access to a pool of scalable resources [3]. As opposed to maintaining their own computing infrastructures, the pay-as-you-go model of cloud computing paradigm enables users to acquire a diverse range of virtual machines with varying flavors (CPU, Memory etc.) for meeting business needs in a more cost effective manner. The flavor of virtualized instances used for executing tasks determines the total execution times of the workflows as well as the associated monetary costs. In order to maximize the achievable cost savings achievable while also ensuring the performance is maintained to a satisfactory level, it is imperative that cost optimized scheduling strategies are designed and implemented.

In particular, the intelligent use of a mix of on-demand and spot instances for workflow executions is a potential means of achieving high cost efficiencies without adversely affecting performance expectations. Spot instances are offered by cloud providers at steep discounts compared to their on-demand counterparts in exchange for reduced reliability. This is because the cloud providers utilize spare computing capacities available for provisioning spot instances, and therefore when the capacity is needed back, the instances are interrupted. Furthermore, as opposed to on-demand instances with fixed prices, the prices of spot instances are not guaranteed to be fixed, as the pricing is dependent on long term supply and demand. The possibility of interruptions and pricing variations adds a layer of complexity that needs to be efficiently handled for enjoying the cost savings without compromising the underlying business requirements. Therefore, it is imperative to establish the right balance between the use of on-demand and spot instances for workflow executions in cloud computing environments.

The ability of Reinforcement Learning (RL) agents to operate in stochastic environments, and learn through experience to act in an optimal manner amid highly dynamic conditions and uncertainties makes it an ideal candidate for overcoming the aforementioned challenges. While many heuristics and meta-heuristics have been proposed for cost optimized workflow scheduling, only very few works have explored the potential of RL in this area. In particular,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CTCNet 2024, July 26–27, 2024, Chengdu, China

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0960-9/24/07

<https://doi.org/10.1145/3685767.3685780>

Deep Reinforcement Learning (DRL) [16] has emerged as an efficient means of solving highly complex problems as evidenced by the recent successes achieved by DRL agents in complex control tasks in fields such as robotics, autonomous driving, healthcare and so on. In this work, we leverage the advanced capabilities of DRL for designing a cost optimized workflow scheduling framework.

The design of action space is a fundamental characteristic of a DRL based formulation of a problem. The action spaces of a vast majority of scheduling problems that are modeled as DRL problems, include a flat set of actions. The action space may be discrete or continuous, and the agent selects an action from the action space. In this work, we propose a novel hierarchical way of designing the action space of the DRL model such that there is a clear distinction between on-demand and spot instances in action selection. A DRL framework comprising multiple actor networks guided by a common critic network is then designed to select a combination of actions from the hierarchical action space, to optimize cost of workflow executions.

Container orchestration engines such as Kubernetes can seamlessly operate atop highly distributed and heterogeneous infrastructures and abstract away the complex coordination details from users. This in turn has enabled users to conveniently deploy workloads across a variety of cloud deployments ranging from private and public clouds to hybrid combinations of these. Complementary frameworks such as Argo workflow engine have emerged to extend the functionalities of Kubernetes to facilitate the management of more complex workloads such as Workflows. The schedulers of these frameworks are pre-configured to follow basic scheduling policies such as bin-packing. These simple policies are not capable of satisfying the complex cost optimization requirements of users. In order to achieve complex user-defined goals it is imperative to incorporate more advanced scheduling policies in the aforementioned workflow management engines. These policies should be capable of adapting to highly stochastic conditions that are inherent in clusters deployed in cloud computing environments. In this regard, we present an end-to-end means of training and deploying the DRL agent proposed in this work in the Argo workflow engine.

More specifically, the following summarizes the main contributions of this work:

- A DRL model for cost optimized scheduling of workflows in a cloud computing environment with the use of a balanced mix of on-demand and spot instances.
- A logical organization of the cluster in a hierarchical manner, along with a novel representation of the action selection process as a tree structure.
- A RL framework with multiple actors guided by a single critic network trained with Proximal Policy Optimization (PPO) algorithm for learning to schedule workflows in the cluster.
- An end-to-end means of training and deploying the proposed DRL agent in a workflow engine. To the best of our knowledge, this is the first attempt at embedding an intelligent agent in an open source container-native workflow engine.

2 RELATED WORK

The use of spot instances for cost optimized workflow scheduling has been studied in a number of studies [8, 10, 19]. However, the methods proposed in some of these works are associated with bidding strategies [9, 19] that are of little relevance in current market, since major cloud providers such as Amazon Web Services (AWS) have devised new pricing models that simplifies the purchasing process of spot instances [1]. Accordingly, users are no longer required to analyze historical price trends and employ strategies for determining maximum bid prices.

In [18], a joint cost and makespan optimization algorithm for workflow executions in cloud is proposed. Authors integrated the popular heterogeneous earliest finish time (HEFT) heuristic with fuzzy dominance sort technique for designing the proposed list scheduling algorithm. [2] also combines the HEFT heuristic with Ant Colony Optimization (ACO) technique for optimizing the same objectives. [17] proposed a makespan and cost aware scheduling technique for hybrid clouds. A combination of Dynamic Voltage and Frequency Scaling (DVFS) and approximate computing is used in [13] for energy efficient and cost optimized workflow scheduling in cloud computing environments.

In [6], authors incorporate artificial neural network with the NSGA-II algorithm for optimizing a combination of objectives associated with workflow scheduling in cloud computing environments. In [15], Zhou et. al. proposed optimization framework for HPC applications deployment on clouds in cost-efficient manner. They leveraged cloud spot market resources with the goal of minimizing application cost while ensuring performance constraints.

In [14], a deep Q learning based multi-agent deep reinforcement learning technique is proposed for optimizing cost and makespan of workflow scheduling in cloud. The work models multi-agent collaboration as a Markov game with a correlated equilibrium, so that the makespan and cost agents are not motivated to deviate from the joint distribution in a unilateral manner. H. Li et. al [7] proposed a weighted double deep Q network based reinforcement learning method for cost and makespan optimized workflow scheduling in cloud environments. Scheduling process includes two levels, in the first level a task is selected from amongst all ready tasks. A pointer network is used for efficiently handling the variable length of the input state. In the second level a VM is selected for executing the selected task. A separate sub agent with a separate reward is used for each objective at each level of the scheduling process. Y. Qin et. al [11] used Q learning for minimizing makespan and energy consumption of workflow executions while adhering to a budget constraint.

3 PROBLEM FORMULATION

The objective of the scheduling framework is minimizing the monetary cost of workflow executions, while also minimizing the execution times. The resource requirements in terms of CPU and memory and the dependencies of workflow tasks are included in the submitted workflow specifications. In the workflow specification submitted by users, a workflow is represented by a DAG, $G = (V, E)$ where the nodes, $V = \{v_0, v_1, \dots, v_n\}$ of the DAG represent tasks of the workflow, and the edges, $E = \{(v_i, v_j) | v_i, v_j \in V\}$

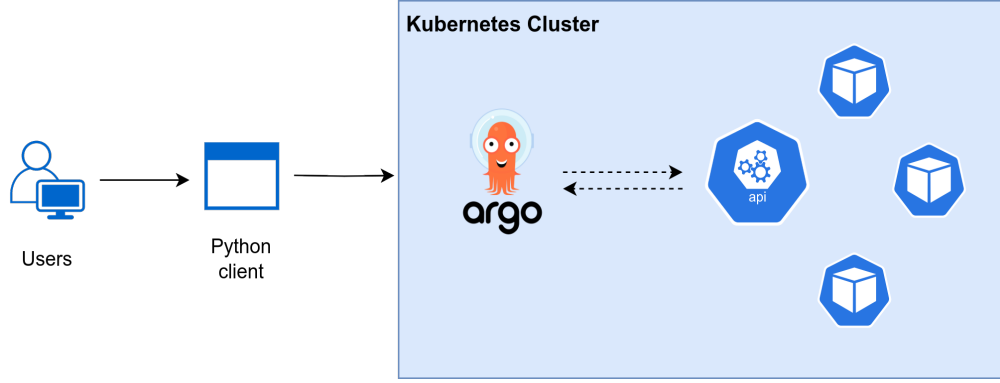


Figure 1: System Architecture

of the DAG represent precedence constraints between tasks. The computation time of a task, t_j can be represented as:

$$CT(t_j) = \frac{L(t_j)}{F} \quad (1)$$

where $L(t_j)$ is the size of task, t_j and F is the processing rate of the node to which is it assigned. All the precedence constraints of task, t_j must be satisfied before its execution commences. Accordingly, the execution of all the predecessors must be completed, and the output data required for the execution of t_j must be transmitted to the node in which it is scheduled. If t_i is an immediate predecessor of t_j and the size of data to be transferred from t_i to t_j is $D(t_i, t_j)$, then the total transmission time (TT) can be denoted as follows:

$$TT(t_i, t_j) = \frac{D(t_i, t_j)}{B} \quad (2)$$

where B is the bandwidth between the execution nodes of t_i and t_j . Task execution delay, $TD(t_j)$ primarily depends on the computation time, $CT(t_j)$ of the task, and the maximum data transfer time from predecessor nodes, $\max_{t_i \in pred(t_j)} TT(t_i, t_j)$. The waiting time, $WT(t_j)$ before a task gets scheduled also contributes to total execution delay. Accordingly, $TD(t_j)$ can be represented as:

$$TD(t_j) = CT(t_j) + WT(t_j) + \max_{t_i \in pred(t_j)} TT(t_i, t_j) \quad (3)$$

The finish time, $FT(t_j)$ of task, t_j that started execution at time, $ST(t_j)$ can then be expressed as:

$$FT(t_j) = ST(t_j) + TD(t_j) \quad (4)$$

The completion time, MT of a workflow is equivalent to the time at which that last task of the workflow completes execution. It can be denoted as:

$$MT = \max_{t_j \in T} (FT(t_j)) \quad (5)$$

where T represents the set of all tasks of the workflow.

The computation cost of t_j that executes in a Node with unit cost per second, UC can be represented as:

$$CC(t_j) = CT(t_j) * UC \quad (6)$$

The cost of execution, MC of a workflow is equivalent to the sum of execution costs of all tasks, and it can be denoted as follows:

$$MC = \sum_{t_j \in T} CC(t_j) \quad (7)$$

The objective of the scheduling problem is to minimize the cost of workflow executions, and it can be denoted as follows:

$$\text{Minimize: } \sum_{i=1}^N MC_i \quad (8)$$

where N is the total number of workflows submitted to the system.

4 BACKGROUND AND PROPOSED APPROACH

In this section, we present a background of the popular container orchestration engine Kubernetes and the open-source Argo workflow engine along with details on how the proposed DRL framework is implemented in the Argo Workflow engine that runs atop the Kubernetes cluster. Worker nodes of the Kubernetes cluster are Virtual Machines with different flavors (compute, memory, and storage capacity of VM instances). Argo workflow engine is deployed in the Kubernetes cluster for the management of workflows submitted by users. The scheduler is responsible for selecting the VMs in which the Pods corresponding to each task of the workflow will be scheduled. A high level architecture of the system is shown in Figure 1. A sequence diagram indicating integration between key components as implemented is shown in Figure 3.

4.1 Kubernetes

Kubernetes is a popular open-source container orchestration engine that facilitates containerized applications to be deployed, scaled, and managed in an automated manner. With Kubernetes, containerized workloads can be conveniently deployed and managed in any infrastructure including public clouds and on-site deployments, as well as hybrid combinations of these as required. Workloads can be seamlessly deployed across multi-cloud environments thus enabling the selection of the most appropriate infrastructure for the execution of different parts of the workload. Furthermore, it facilitates the up-scaling and down-scaling of clusters to suit demand

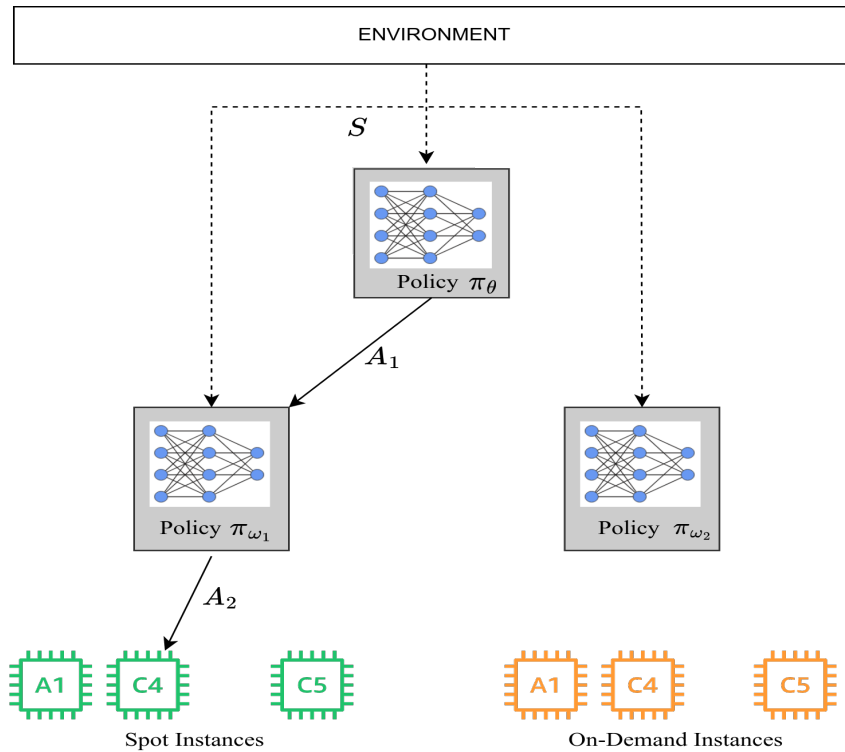


Figure 2: Proposed hierarchical action space and multi-actor DRL model

variations of applications, which in turn helps reduce costs due to reduced resource wastage. The need for manual intervention is minimized since Kubernetes monitors the health of the deployment and redeploys new containers in the event of a failure to restore operations, and this helps reduce application downtime. Owing to the multitude of benefits offered by Kubernetes, it has become the defacto platform for the deployment and management of containerized workloads. In this work, we extend the capabilities of the default Kubernetes scheduler by incorporating intelligence into it with the use of RL techniques.

A Kubernetes cluster consists of a set of virtual or physical machines which are referred to as Nodes. The smallest unit deployable in Kubernetes is referred to as a Pod. Pods are hosted by Nodes. A Pod may comprise one or more tightly coupled containers that share storage and network resources, it also contains a specification of how the containers are to be run. The contents of a Pod run in a shared context, and are always located and scheduled together. Pods and Nodes of a Kubernetes cluster are managed by the control plane. It comprises multiple components that work together for managing the cluster. Kube-API server exposes the Kubernetes API that serves as the front end of the Kubernetes control plane. Cluster data are stored in a key-value store termed etcd. The kube-controller-manager runs several controller processes that monitor and regulate the cluster state. Cloud-controller-manager handles cloud-specific control logic. Kube-scheduler is responsible for scheduling unassigned Pods to Nodes for execution.

4.2 Argo Workflow Engine

Argo workflow engine is an open-source container-native workflow engine that facilitates the orchestration of workflows on Kubernetes. Argo workflows are implemented as a Custom Resource Definition (CRD) in Kubernetes. This enables Argo workflows to be managed using kubectl and they integrate natively with Kubernetes services including secrets, volumes and Role Based Access Control (RBAC).

The workflow engine comprises two main components: the Argo server and the workflow controller. The Argo API is exposed by Argo server and the controller performs workflow reconciliation. In the reconciliation process, the workflows that are queued based on additions and updates to workflows and workflow pods, are processed by a set of worker goroutines. The controller processes one workflow at a time. Both Argo server and controller run in the Argo Namespace.

Each task of workflow results in the generation of a Pod. Each pod includes three containers. The main container runs the image that the user has configured for the task. The init container is an init container that fetches artifacts and parameters and makes them available to the main container. Wait container performs tasks related to clean up including the saving of artifacts and parameters.

Argo provides multiple templates for defining workflow specifications and dependencies. For example, a workflow can be defined as a sequence of steps. Alternatively, DAGs can be used for defining a workflow and its dependencies. As this facilitates the representation of complex workflows and parallelism, in this work we have used DAGs for modeling workflows.

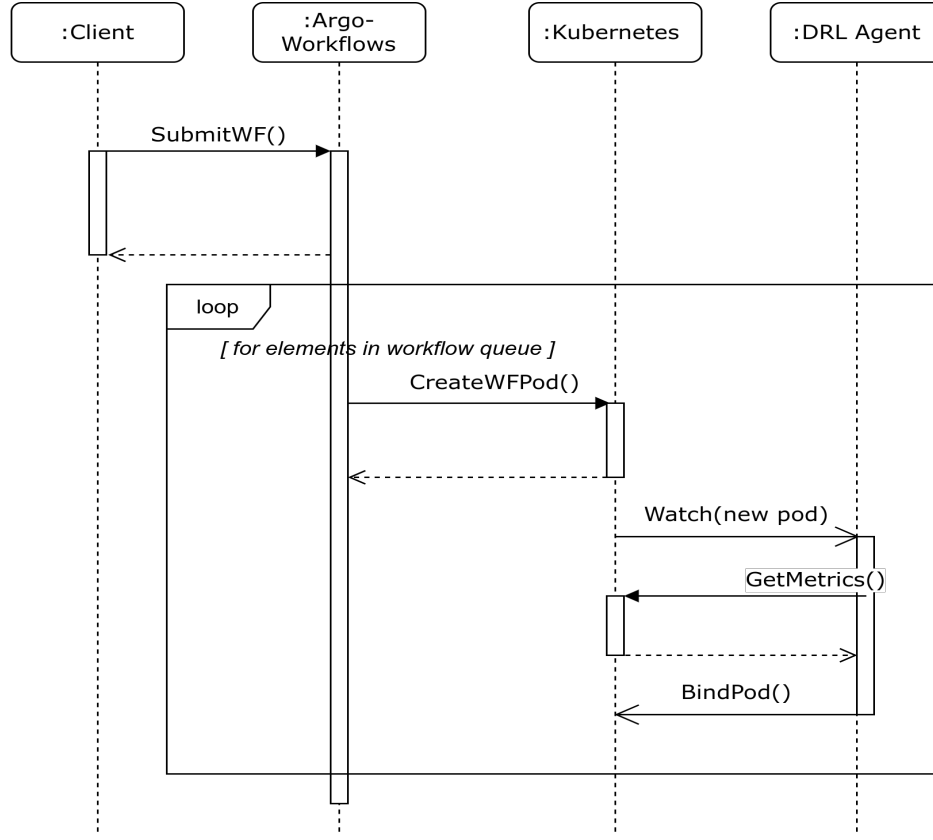


Figure 3: Sequence diagram of DRL based scheduling framework

A workflow specification comprises a set of Argo templates, each with an optional input section, an optional output section, and either a list of steps where another template is invoked by each step or a container invocation (leaf template). The options accepted by the container section of the workflow specification are the same options as the container section of a Pod specification.

4.3 Reinforcement Learning

In the RL paradigm, an agent learns in a trial-and-error manner by interacting with the environment. The agent receives a *reward*, r_t when it performs an *action*, a_t in a particular *state* s_t , and then the *environment* transitions to the next state, s_{t+1} . The process repeats until the agent encounters the *terminal state* at which point the *episode* terminates. Markov Decision Process (MDP) can be used for mathematically modeling RL problems. According to the Markov property, it is considered the next state of the environment and the reward received depends solely on the current state and the agent's action in the current state. The cumulative discounted rewards, G_t at any given timestep, t is expressed as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (9)$$

where γ is a discount factor and $\gamma \in (0, 1)$. The RL agent operates with the goal of maximizing the expected return, $E[G_t]$ from each

state, s_t . A *policy*, $\pi(a_t|s_t)$ is a mapping from the current observation of the environment to a probability distribution of the actions that can be taken from the current state. During the training process, a traditional RL agent is required to visit all the states of the problem and store experiences in space-consuming tabular formats. This is a limitation that makes it infeasible to apply the traditional RL paradigm to problems with high dimensional states and action spaces. The integration of Deep Learning with the RL paradigm gave rise to an efficient means of overcoming the aforementioned limitation through the use of neural networks as function approximators for enabling the agent to estimate the value of a state or an action when it encounters a similar circumstance. In the resulting Deep Reinforcement Learning (DRL) paradigm, the policy, $\pi(a_t|s_t)$ is modeled as a parameterized function, $\pi_{\theta}(a_t|s_t)$ where θ is an adjustable parameter derived with an RL algorithm.

In *value based* RL methods, the RL agent attempts to learn a state-value function, $v_{\pi_{\theta}}(s)$, or a state-action value function, $Q_{\pi_{\theta}}(s, a)$. As the name implies, the state-value function estimates the value of a state, and it can be expressed in terms of expected return when following a policy π_{θ} starting from the state, s as shown in Equation 10. Equation 11 indicates the state-action value function which is the expected return when action, a_t is taken at state, s_t , and policy, π_{θ} is followed afterward.

$$v_{\pi_{\theta}}(s) = E_{\pi_{\theta}}[G_t | s_t = s] \quad (10)$$

$$Q_{\pi_{\theta}}(s, a) = E_{\pi_{\theta}}[G_t | s_t = s, a_t = a] \quad (11)$$

In *policy gradient* RL methods, the agent directly learns the policy, $\pi_{\theta}(a_t | s_t)$. Typically gradient-based techniques on the expectation of returns are used for learning the policy. Equation 12 indicates the form of the most commonly used gradient estimator.

$$\hat{g} = \hat{E}_t[\nabla_{\theta} \ln \pi_{\theta}(a_t | s_t) \hat{A}_t] \quad (12)$$

where, \hat{A}_t is an estimator of the advantage function at timestep, t and π_{θ} is a stochastic policy. In an RL algorithm that alternately performs sampling and optimization, the expectation $\hat{E}_t[\dots]$ indicates the empirical average computed over a batch of samples. For evaluating the performance of the policy, a performance objective the gradient of which is the policy gradient estimator, \hat{g} is defined. Accordingly, \hat{g} is obtained by differentiating the objective:

$$L^{PG}(\theta) = \hat{E}_t[\ln \pi_{\theta}(a_t | s_t) \hat{A}_t] \quad (13)$$

Although multiple rounds of optimizations can be performed on the loss, $L^{PG}(\theta)$ defined in Equation 13 using a single trajectory of experience samples, it is not desirable since that could lead to adverse consequences such as policy updates that are destructively large. In order to overcome the aforementioned issue, in Proximal Policy Optimization [12] method, a clipped surrogate objective is used. More specifically, the degree to which new policy, $\pi_{\theta}(a_t | s_t)$ is allowed to change from old policy, $\pi_{\theta_{old}}(a_t | s_t)$ is restricted by the use of a clip function as indicated in Equation 14. The clip function, $\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t$ removes the desirability of large policy updates that changes the $r_t(\theta)$ ratio beyond the interval $[1 - \epsilon, 1 + \epsilon]$.

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t)] \quad (14)$$

where $r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)}$

Actor-critic is a branch of RL algorithms that combines the advantages of value-based methods and policy gradient RL methods. The actor is the policy that outputs a probability distribution over the actions that can be taken in the current state, and the critic is the value function approximator that evaluates the actions taken by the actor as per the policy.

4.4 Proposed RL Framework

As previously discussed, the default kube-scheduler takes multiple factors into account in formulating scheduling decisions including resource requirements and constraints, specifications of affinity and anti-affinity, deadlines, and interference caused by co-located workloads. These policies need to be pre-defined and may suffer from the general limitations of heuristic scheduling techniques. In this work, we override the default behavior and incorporate intelligence into the scheduler by training a DRL agent to select appropriate scheduling decisions with the objective of achieving a desired goal.

4.4.1 Agent Environment. The problem of scheduling workflows in a cloud cluster can be simplified by formulating it as a dependent task-scheduling problem. In the Argo workflow engine, pods corresponding to independent tasks are scheduled directly in the cluster

for execution, while the tasks with dependencies are not scheduled until the parent tasks have completed execution. Whenever the workflow scheduler (RL agent), discovers a pod that is not assigned to a node, it takes the current state of the environment as input and outputs the most desirable node for task execution based on the trained policy. The environment then transitions to the next state. Accordingly, the timesteps of the proposed RL model are discrete and event-driven. The state, action, and reward of the RL model are designed as follows:

State Space: State of the environment comprises of total CPU and Memory requirements of the task, and nodes together with the estimated waiting time at each node based on the number of pods executing in each node.

Action Space: Compared to the problem of scheduling tasks in a cluster comprising nodes from the same cloud data center, scheduling tasks in a multi-cloud cluster is more challenging since resource capacities and cost are not the only factors that differentiate nodes. In such scenarios, the intercloud communication delay is an important factor that needs to be factored into the formulation of scheduling decisions. This requirement is further heightened in workflow scheduling due to the presence of data dependencies among tasks that may result in costly data transfers if communication costs among nodes from different clouds are ignored.

In the most straightforward design of the action space, the action of selecting any one of the nodes in the multi-cloud cluster can be represented together in a flat action space. In this approach, the burden of distinguishing nodes from different clouds lies with the DRL agent. Although the agent may eventually manage to learn the presence of nodes from multiple clouds based on rewards and thereby develop an internal representation of the multi-cloud composition of the nodes, it will inevitably reduce the training efficiency of the agent. Furthermore, as the size of the cluster grows, flat action spaces are more prone to the problem of the 'curse of dimensionality'.

In order to efficiently overcome the aforementioned challenges, we have designed the action space considering a logical organization of cluster. In the logical organization, nodes from different pricing categories are grouped together as shown in Figure 2. Accordingly, we define a hierarchical action space for the problem as follows:

$$A = \{(a_1, a_2) | a_1 \in \{\pi_{\omega_1}, \pi_{\omega_2}\} \ \& \ a_2 \in \{1, 2, \dots, N_{a_1}\}\} \quad (15)$$

where N_{a_1} is the total number of nodes in the cluster that belong to the group given by action a_1 . The action, a_1 corresponds to the selection of a node group, and the action, a_2 corresponds to the selection of a node from the group. An action at each timestep then corresponds to the joint action (a_1, a_2) .

Reward: Reward is the estimated cost of execution at the allocated node computed with Equation 6.

4.4.2 Multi-Actor RL Algorithm. The hierarchical action space described above can be represented as the tree structure in Figure 2. Each level of the tree corresponds to an action selection sub-problem. The first level of the tree represents the sub-problem of selecting a node group and the second level represents the sub-problem of selecting a node. We then adopt the hybrid actor-critic

Table 1: Resource configurations of Kubernetes cluster

Instance Type	CPU Cores	Memory(GB)	Quantity		Price	
			Spot	On-demand	Spot	On-demand
t4g.large	2	8	2	2	\$0.033/h	\$0.0672/h
t4g.xlarge	4	16	3	2	\$0.0857/h	\$0.1344/h
t4g.2xlarge	8	32	1	1	\$0.1589/h	\$0.2688/h

Algorithm 1 Actor-Critic based Scheduling Framework with PPO

- 1: Initialize actor networks and critic network with random weights
 - 2: Initialize the training parameters: α, β, γ
 - 3: **for** episode = 1 to N **do**
 - 4: Reset the environment
 - 5: **for** step = 1 to T **do**
 - 6: Input the state of the environment to actor networks
 - 7: Select action a_1 from π_θ
 - 8: Select action a_2 from π_{ω_i}
 - 9: Execute the combined action (a_1, a_2) and observe the corresponding reward r_t and next state of the system s_{t+1}
 - 10: Store the most recent transition (s_t, a_t, r_t, s_{t+1}) in memory D
 - 11: Compute advantage estimates \hat{A}_1 to \hat{A}_T
 - 12: **for** $j = 1$ to K **do**
 - 13: Randomly sample a mini-batch of samples of size S from D
 - 14: **for** $p = 1$ to S **do**
 - 15: Update critic network:

$$\sigma \leftarrow \sigma + \beta \delta_t \nabla v_\pi(s_t | \sigma)$$
 - 16: Update first actor network:

$$\theta \leftarrow \theta + \alpha \hat{A}_p \nabla \ln \pi(a_1 | s, \theta)$$
 - 17: Update second actor network:

$$\omega \leftarrow \omega + \gamma \hat{A}_p \nabla \ln \pi(a_2 | s, \omega)$$
 - 18: Clear memory D
- return**

technique presented in [5] for selecting joint actions from the hierarchical action space. Different from a traditional actor-critic algorithm which contains a single actor-network and a single critic network, in the proposed architecture multiple parallel actor networks are guided by a common critic network.

As shown in Figure 2 each action-selection sub-problem is handled by a separate actor network. Accordingly, one actor network learns a stochastic policy for selecting a node group. For each of the node groups, a separate actor network learns a stochastic policy for selecting a node from the respective node group. The critic network estimates the state value function, $V(s)$. The advantage function provided by the critic network is used for updating the stochastic policies. Actor networks are separately updated at each timestep by their respective update rules. We used the PPO method for updating the networks. Algorithm 1 summarizes the steps included in the training process of the DRL agent.

5 PERFORMANCE EVALUATION

In this section we present the details of the experimental testbed used for evaluating the proposed DRL framework along with the results of the performance evaluation.

5.1 Experimental Testbed

The resource configurations and composition of the Kubernetes cluster is shown in Table 1. Argo workflow engine is installed in the cluster in a separate namespace. A Python client that communicates with Argo API server was developed for submitting workflows and querying about the execution statistics of workflows.

5.2 Experimental Dataset

The experimental dataset comprises of a set of Map-Reduce workflows. Each map task performs a CPU intensive parallelizable computation that involves finding the sum of the square-roots of numbers in a given input range. Experiments were conducted at different arrival rates drawn from a uniform distribution. Experiments were also conducted at different task sizes and parallelism levels.

5.3 DRL Scheduler Implementation

The Argo workflow engine uses the default Kubernetes scheduler for allocating tasks (i.e. pods) to nodes. We have overridden it with a DRL agent trained according to the proposed DRL framework. The configurations of all test workflows were updated such that they are scheduled with the custom DRL scheduler instead of the default scheduler. Keras library [4] was used for developing the DRL framework.

Kubernetes metrics server collects resource metrics of the underlying nodes from Kubelets and shares it with the Kubernetes API server via the Metrics API. Therefore, by querying the Kubernetes API server we were able to retrieve near real-time CPU and Memory usages of the nodes, which were required to formulate the state space composition that needs to be provided as the state of the environment to the agent at each timestep of the episode. At the end of each episode, the python client queries the Argo API server for retrieving the execution statistics of workflows including resource times, start and end times of workflows and success rates which are then used for computing the resource usages and associated costs. The client also queries Kubernetes API server for retrieving node metrics that is required for computing the up times of nodes.

5.4 Comparison Algorithms

The performance of the proposed DRL algorithm was compared against three scheduling policies. **Random** policy allocates tasks to nodes in a Random manner, and is completely agnostic to pricing as

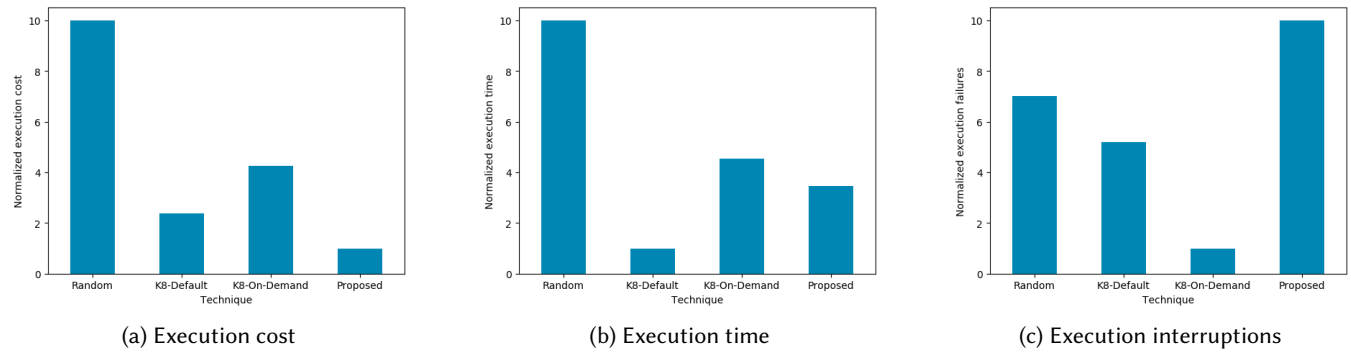


Figure 4: Comparison of performance of scheduling algorithms on an experimental dataset

well as other resource utilization levels of the cluster. **K8-Default** refers to the default scheduling policy of Kubernetes cluster. **On-Demand** is a policy that uses Kubernetes default scheduler but the selection is limited to the on-demand instances.

5.5 Experimental Results

Figure 4a shows the performance of the algorithms on the experimental dataset with respect to monetary cost of workflow executions. Random algorithm has incurred the highest cost owing to the fact that it distributes tasks across multiple instances without trying to optimize resource utilization or cost. In comparison the Kubernetes scheduler exhibits much better cost savings. By default, it is designed to select the most appropriate node through a node filtering and scoring process. In the filtering phase, nodes that are feasible for executing the pod are selected, and then they are ranked according to a scoring process. Based on the outcome of the filtering and scoring process the most appropriate node for pod execution is selected. Clearly, this process has resulted in much better resource efficiency and thereby cost savings in comparison to random allocation. As expected, K8-On-Demand method has incurred a higher cost than the default policy since it is only allowed to make a selection from amongst the on-demand instances which have a higher unit cost. The proposed method has resulted in the highest cost savings. The significant reduction in cost is due to the intelligent cost aware allocation of pods among the instances in the cluster.

Figure 4b shows a comparison of the execution times of workflows scheduled with different algorithms. Again, the highest amount of time is taken by Random algorithm. K8-On-Demand has resulted in higher execution times compared to k8-Default due to the limited selection of instances available for scheduling. K8-Default has resulted in the least execution time since it distributes pods amongst multiple high scoring nodes, without considering the respective unit cost differences. Proposed algorithm has incurred slightly higher cost since it's favoring nodes that are of low cost which leads to more pods being assigned to the same nodes, hence resulting in increased execution times. This is expected since instances with more vCPUS are more expensive, which results in a trade-off between execution time and cost.

Figure 4c shows the number of execution failures. Execution failures in the experimental context are solely due to the interruption of spot instances which leads to workflows timing out and thereby failing to complete. As expected the proposed algorithm results in the highest failures since it is favoring spot instances for task executions, and the spot instances are subjected to interruptions. This is a known trade-off associated with the use of spot instances, therefore it is important to restrict the use of spot instances for failure tolerant workflows.

6 CONCLUSIONS

In this work, we designed a DRL technique for cost-optimized workflow scheduling in Cloud environments by the intelligent use of spot and on-demand instances. We then designed and implemented an end-to-end system for integrating and training the DRL agent in the container-native Argo workflow engine that runs atop Kubernetes. As evidenced by the results of the experiments, higher cost savings can be achieved by overriding the default schedulers with intelligent cost-optimized scheduling policies.

REFERENCES

- [1] [n. d.]. New Amazon EC2 Spot pricing model. <https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing/>.
- [2] Ali Belgacem and Kadda Beghdad-Bey. 2022. Multi-objective workflow scheduling in cloud computing: trade-off between makespan and cost. *Cluster Computing* 25, 1 (2022), 579–595.
- [3] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.* 25, 6 (2009), 599–616. <https://doi.org/10.1016/J.FUTURE.2008.12.001>
- [4] François Chollet et al. 2018. Keras: The python deep learning library. *ascl* (2018), ascl-1806.
- [5] Zhou Fan, Rui Su, Weinan Zhang, and Yong Yu. 2019. Hybrid actor-critic reinforcement learning in parameterized action space. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*. 2279–2285.
- [6] Goshgar Ismayilov and Haluk Rahmi Topcuoglu. 2020. Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing. *Future Generation Computer Systems* 102 (2020), 307–322.
- [7] Huifang Li, Jianghang Huang, Binyang Wang, and Yushun Fan. 2022. Weighted double deep Q-network based reinforcement learning for bi-objective multi-workflow scheduling in the cloud. *Cluster Computing* (2022), 1–18.
- [8] Thanh-Phuong Pham and Thomas Fahringer. 2020. Evolutionary multi-objective workflow scheduling for volatile resources in the cloud. *IEEE Transactions on Cloud Computing* 10, 3 (2020), 1780–1791.
- [9] Deepak Poola, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2014. Fault-tolerant workflow scheduling using spot instances on clouds. *Procedia Computer Science* 29 (2014), 523–533.

- [10] Deepak Poola, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2016. Enhancing reliability of workflow execution using task replication and spot instances. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 10, 4 (2016), 1–21.
- [11] Yao Qin, Hua Wang, Shanwen Yi, Xiaole Li, and Linbo Zhai. 2020. An energy-aware scheduling algorithm for budget-constrained scientific workflows based on multi-objective reinforcement learning. *The Journal of Supercomputing* 76 (2020), 455–480.
- [12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 <http://arxiv.org/abs/1707.06347>
- [13] Georgios L Stavrinides and Helen D Karatza. 2019. An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations. *Future Generation Computer Systems* 96 (2019), 216–226.
- [14] Yuandou Wang, Hang Liu, Wanbo Zheng, Yunni Xia, Yawen Li, Peng Chen, Kunyin Guo, and Hong Xie. 2019. Multi-objective workflow scheduling with deep-Q-network-based multi-agent reinforcement learning. *IEEE access* 7 (2019), 39974–39982.
- [15] Amelie Chi Zhou, Jianming Lao, Zhoubin Ke, Yi Wang, and Rui Mao. 2022. FarSpot: Optimizing Monetary Cost for HPC Applications in the Cloud Spot Market. *IEEE Transactions on Parallel and Distributed Systems* 33, 11 (2022), 2955–2967. <https://doi.org/10.1109/TPDS.2021.3134644>
- [16] Guangyao Zhou, Wenhong Tian, and Rajkumar Buyya. 2024. Deep Reinforcement Learning-based Methods for Resource Scheduling in Cloud Computing: A Review and Future Directions. *Artificial Intelligence Review* 57 (2024). <https://doi.org/10.1007/s10462-024-10756-9>
- [17] Junlong Zhou, Tian Wang, Peijin Cong, Pingping Lu, Tongquan Wei, and Mingsong Chen. 2019. Cost and makespan-aware workflow scheduling in hybrid clouds. *Journal of Systems Architecture* 100 (2019), 101631.
- [18] Xiumin Zhou, Gongxuan Zhang, Jin Sun, Junlong Zhou, Tongquan Wei, and Shiyuan Hu. 2019. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. *Future Generation Computer Systems* 93 (2019), 278–289.
- [19] Behrooz Zolfaghari and Saeid Abrishami. 2022. A multi-class workflow ensemble management system using on-demand and spot instances in cloud. *Future Generation Computer Systems* 137 (2022), 97–110.