Deep Reinforcement Learning for Scheduling Applications in Serverless and Serverful Hybrid Computing Environments

Anupama Mampage, Shanika Karunasekera, *Member, IEEE*, and Rajkumar Buyya, *Fellow, IEEE*

***Abstract*—Serverless computing has gained popularity as a novel cloud execution model for applications in recent times. Businesses constantly try to leverage this new paradigm to add value to their revenue streams. The serverless eco-system accommodates many application domains successfully. However, its inherent properties such as cold start delays and relatively high per unit charges appear as a shortcoming for certain application workloads, when compared to a traditional Virtual Machine (VM) based execution scenario. A few research works exist, that study how serverless computing could be used to mitigate the challenges in a VM based cluster environment, for certain applications. In contrast, this work proposes a generalized framework for determining which workloads are best able to reap benefits of a serverless computing environment. In essence, we present a potential hybrid scheduling solution for exploiting the benefits of both a serverless and a VM based serverful computing environment. Our proposed framework leverages the actor-critic based deep reinforcement learning architecture coupled with the proximal policy optimization technique, in determining the best scheduling decision for workload executions. Extensive experiments conducted demonstrate the effectiveness of such a solution, in terms of user cost and application performance, with improvements of up to 44% and 11% respectively.**

***Index Terms*—Application scheduling, function latency, reinforcement learning, serverful computing, serverless computing, user cost efficiency.**

I. Introduction

WITHServerless computing has attracted the attention of majority of cloud users with its distinguished properties which elevate the efficiency of the entire process of application deployment and their subsequent operations. As a result, many new applications are designed to suit the stateless, short running nature of serverless functions, for easy transition in to this novel computing paradigm. Nevertheless, even when an application in its design is fully compatible with a serverless architecture, the nature of the application workload may render it unsuitable for execution in this environment. A key feature of any serverless platform is its ability to auto-scale the deployed application at a very granular level, with scaling to zero at its extreme [1]. Achieving such optimal resource efficiency is paramount to the successful operation of this provider managed multi-tenant computing platform. This distinct function level auto-scaling property is established by closely following the demand patterns and scaling up and down function resources just-in-time as required. With such an adhoc scaling policy, the occurrence of certain delays in user request executions due to time taken for resource creation known as cold start delays, is unavoidable. While for some applications, this occasional and rather small delay could be deemed insignificant, a latency sensitive application with a very short runtime could face detrimental effects from such unprecedented delays [2].

Serverless platforms charge their users only for the resources consumed for application execution, calculated with a millisecond accuracy [3]. To the service provider this may incur a loss at times when load levels tend to be irregular, since their infrastructure would need to stay alive throughout, including the idling periods. In order to compensate for this potential unrecoverable cost, it is observed that the per unit billing rates for serverless services are considerably higher compared to traditional VMs [2]. Thus, if an application sustains a constant high level of traffic, the feasibility of using a serverless deployment need to be studied. However, if the load levels are irregular with sudden bursts of traffic, using serverless functions and paying for only the resource consumed time would be understandably cheaper.

On the other hand, a VM when rented out, could be used for a longer period without facing adhoc resource creation times. As long as the application is having a regular high traffic level to keep the resources busy, the usage of a VM based set up could prove to be much more cost effective compared to a serverless execution [4]. In contrast, if load fluctuations are high with long periods of little or no traffic, maintaining a VM resource would not be as meaningful cost-wise. An on-off mechanism for VMs is also not viable considering the relatively high start up times.

In addition to the initial decision on determining the execution platform, the subsequent decision on choosing a host node on the selected platform too has implications on both time and cost factors for the users [5]. While on the serverless platforms, choosing a host node with warm function instances saves up on request waiting times, careful load balancing on Infrastructure-as-a-Service (IaaS) clusters is directly related to maximizing rented resource utilization and in turn earning cost savings.

Received 19 November 2023; revised 3 November 2024; accepted 5 December 2024. Date of publication 7 January 2025; date of current version 10 April 2025. This work was supported in part by the Australian Research Council. *(Corresponding author: Rajkumar Buyya.)*

The authors are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, VIC 3052, Australia (e-mail: mampage@student.unimelb.edu.au; karus@unimelb.edu.au; rbuyya@unimelb.edu.au).

Digital Object Identifier 10.1109/TSC.2024.3520864

Considering the aforementioned facts, it is useful to be able to understand the workload patterns for an application and come up with a suitable schedule for executing user requests, targeting both time and cost effectiveness. A few existing research works have initiated the first steps in this regard, mostly by exploring how serveress computing can be used as an add-on to mitigate various shortcomings of a VM based serverful deployment [6], [7], [2]. The majority of these works target specific application scenarios such as web-services and High Performance Computing (HPC) workloads and try to determine at which point, a switch to a serverless execution would be useful. In our work, we aim to provide a fully generalized intelligent solution for request scheduling, which extracts the best in both a serverless and a serverful infrastructure. Our proposed framework is capable of determining not only the deployment environment, but also the specific resource in that environment that is ideal for running a particular user request considering both application performance and user cost. A fully automated hybrid framework such as this could be a potential offering by cloud service providers which would have many use cases.

Deep Reinforcement Learning (DRL) is very popular among researchers nowadays for solving cloud resource management related problems due to its experience based learning strategy which is proven to be effective in exploring dynamic cloud computing scenarios. Our proposed solution employs an actor-critic architecture enhanced with a hierarchical action space which first determines the ideal deployment environment after which the most suitable cluster node is selected. The key **contributions** of our work are as follows:

1) We formulate the problem of scheduling an application request on a serverless and serverful hybrid cluster environment, based on RL.
2) We propose a novel actor-critic architecture with a hierarchical action space which is capable of decision making at two levels. The mode of deployment is decided in the first level while the node for scheduling within the selected cluster is decided in the second. The DRL agent is trained to reach the best optimization levels in terms of application performance and user cost for a given workload.
3) The DRL agent is modeled to capture application workload as well as the serverless and serverful cluster resource details and behavioral patterns in order to gain a comprehensive understanding on its action environment.
4) We evaluate and compare our approach with baseline scaling techniques using real world applications, together with function traces captured from Microsoft Azure Functions.

The rest of the paper is organized as follows: Section II highlights existing relevant literature. Section III describes our system model and presents the mathematical formulation of our problem. Followed by this, Section IV describes the DRL based hybrid scheduling framework. Section V presents the details of the DRL agent training environment and discusses the performance of our proposed solution. Finally, Section VI explains plans for future work.A list of notions used in the paper are listed in Table II.

II. Related Work

A. Serverless and Serverful Hybrid Scheduling

The concept of utilizing a hybrid serverless and VM based environment for application execution is still at its inception. A few works exist in literature which have attempted to explore this hybrid approach for various use cases.

[2] study how a serverless deployment could help alleviate shortcomings of VM auto-scaling for Machine Learning (ML) inference services. They propose a framework which uses serverless functions whenever VMs with free resources are not available. The required VM instances are then spawned based on either a reactive or predictive scaling policy. Load balancing on VMs is done using a bin-packing method. However, they use dedicated VM clusters for serving different ML models and the heuristics used for execution node selection are not ideal for optimizing user cost. Another approach of using cloud functions for interim processing while VMs are being launched is discussed in [4]. [7] present a system to dynamically switch a micro-service deployment between functions and VMs. They mainly focus on resource contention that could occur among serverless functions and use input from a contention monitor when taking a decision to switch an application load from IaaS based deployment. They handle resource scaling by always directing traffic to one deployment mode and reactively creating the required resources on the other. With highly dynamic workload patterns, this may not result in the ideal usage of resources in both platforms. A time series analysis and a classification algorithm is used in [8] for deciding the best deployment environment for a given time range. An initial study on determining for which workload scenarios, a hybrid deployment approach would be beneficial is conducted in [9]. A solution for leveraging serverless computing for executing HPC workflows is presented in [6]. In order to determine which environment is ideal for running each task, they run everything on VMs and then on a serverless platform. The I/O overhead and the execution time on each platform are taken in to consideration for decision making. Cost efficiency is not included in the scope of their work. They suggest a heuristic for mitigating container cold-start delay, but do not account for VM start up delays.

B. Serverless Resource Management With RL

RL has been used successfully in a number of research works in recent times for enhancing resource management in serverless computing environments. The provided solutions mostly address function scheduling, scaling problems or used for determining optimum resources allocations for a function.

A multi-step DQN solution and a policy gradient algorithm are discussed in [10] and [11] for determining the ideal cloud based host node for running a serverless function. A number of DRL based frameworks are presented in [12], [13], [14], and [15] for serverless cloud-edge computing environments. Q-Learning solutions are presented in [16], [17] and [18] for horizontally scaling serverless functions. Further, DRL and RL implementations are also proposed for combined horizontal and vertical scaling decision frameworks in [19] and [20].

TABLE I
SUMMARY OF LITERATURE REVIEW

Work	Decision Level		Technique	Decision Parameters					Generalizability	VM Heterogeneity
	Deployment Mode	Scheduling Node		Optimization Objective		Container & VM Cold Start Awareness	Workload Awareness	Overall System Awareness		
				Response Time	User Cost					
[2]	✓		Heuristic/Linear Regression	✓	✓	✓	✓			✓
[4]	✓		Heuristic	✓	✓	✓	✓			
[7]	✓		Queueing Theory/Heuristic	✓	✓	✓	✓		✓	
[8]	✓		ML	✓	✓		✓			
[6]	✓		Heuristic	✓			✓		✓	✓
Our proposed work	✓	✓	DRL(A2C-PPO)	✓	✓	✓	✓	✓	✓	✓

TABLE II
DEFINITION OF SYMBOLS

Symbol	Definition
N	Set of nodes in the serverless cluster
V	Set of VMs in the serverful cluster
Q	Total number of deployed functions
$n^c_{i(t)}$	Available CPU in i^{th} node, $i \in [1, Q]$
$n^m_{i(t)}$	Available memory in i^{th} node, $i \in [1, Q]$
$v^c_{l(t)}$	Available CPU in l^{th} VM, $l \in [1, M]$
$v^m_{l(t)}$	Available memory in l^{th} VM, $l \in [1, M]$
R_j	The j^{th} request of an application, $j \in [1, B]$
C_j	Container running the j^{th} request of an application
C^c_j	Requested CPU by the j^{th} container
Cj^m	Requested memory by the j^{th} container
R^{r0}_j	Standard response time of the j^{th} request
R^r_j	Actual response time of the j^{th} request
E_j	Execution time of request, E_j
W	Per MB/s charge for serverless executions
L	Charge per request for serverless executions
p_i	Unit price of VM, v_l
t_l	Total active time of VM, v_l

A summary of related works in literature, which explore the space of serverless and IaaS cluster hybrid scheduling, is provided in Table I. The existing studies are compared in terms of their provided solution scope, used technique, the objectives of optimization, awareness on various system parameters, generalizability (adaptability for multiple application workloads) and consideration for VM heterogeneity in the serverful cluster.

III. HYBRID SCHEDULING

A. System Model

Our system model is primarily composed of two service clusters, one for serverless deployments and one for a serverful (IaaS) deployment. A global load balancer and a resource manager component handles the forwarding of user requests to one of the clusters for execution. Fig. 1 illustrates the system model of our proposed hybrid execution engine.

User requests are received at the hybrid load balancer as shown in the diagram. This global load balancer is the decision making body which outputs the best deployment mode/environment and also the specific scheduling node for an application request. Articulating the functionality of this novel functional component is the focus of this work. The hybrid resource manager is a database server which periodically derives and updates the resource as well as behavioral metrics of both the service clusters.

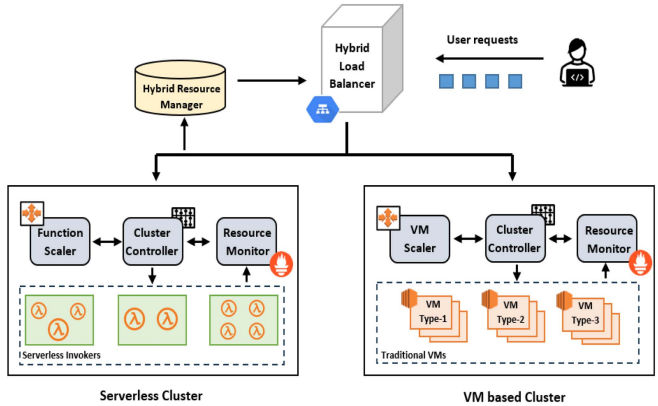


Fig. 1. The system model of the hybrid application execution environment.

The serverless cluster is comprised of a set of invokers which host the containers for function execution. These could be servers or VMs, and referred to as 'nodes' from here onwards in the paper. The cluster controller acts as the governing body which coordinates the communication and overlooks the actions of all the other functional components. It is also the entry point to the cluster. Any new request that is decided to be deployed in a serverless environment, is received at the controller along with the selected node information for scheduling the same. The resource monitor is a monitoring tool which scrapes cluster metrics including resource details of nodes and function containers and also the performance data of the requests in execution and passes on to the global resource manager. Upon receipt of a new request, the controller checks if the selected node contains warm function instances for the request type. If so, the request is forwarded to an existing ready container. In case such idling instances are not available, the function scaler spawns a new container in the preferred node and deploys the function code along with any dependencies. Then the request is forwarded to the new instance. This results in a 'cold start' delay for the request. To have more warm instances in order to minimize this delay, the scale-in process allows a set idle time for the resource, once a request finishes its execution. The function executions are charged as per the billing model in commercial serverless platforms, i.e. at a GB-second rate with a millisecond (ms) granularity in addition to a per request rate.

The IaaS cluster is composed of rented VM resources with varying cpu and memory capacities, from a cloud provider under the IaaS model. We consider resources similar to on-demand EC2 VMs from Amazon [21], which are recommended for

short term, unpredictable workloads that cannot be interrupted. Users are charged per second of usage derived from the hourly rate. The resource monitor gathers cluster metrics similar to that in the serverless cluster. The VM scaler is equipped with a cpu-threshold based scaling policy following Amazon EC2 auto scaling [22]. Accordingly, VMs are scaled out in order to maintain the average cpu utilization at the given threshold. Once a VM is freed after all the executions, it is scaled-in only after staying idle for a set time duration, so that the frequency of cold starting instances is minimized. We define a maximum cluster size, and maintain a record of the VMs that are active, stopped, and pending creation at a given time. Once a request arrives at the controller with an associated VM id, if the particular resource is already running, the request is forwarded to it. If it is pending creation, the request is queued until the resource comes alive.

B. Problem Formulation

Consider $N = \{n_1, n_2, \ldots, n_Q\}$ to be the set of nodes in a serverless computing environment. Each node has a cpu and memory capacity measured in terms of vCPU cores and Mega Bytes (MBs). The unallocated, available cpu and memory values of node n_i, $1 \leq i \leq Q$ at time t is $n^c_{i(t)}$ and $n^m_{i(t)}$ respectively. C_j is the container running the j^{th} ($1 \leq j \leq B$, B being the total number of requests for the function) request of an application/function deployed in the cluster. Following resource demand and capacity constraints need to be satisfied for C_j to be deployed on node n_i at time t.

$$C^c_j \leq n^c_i(t), \quad C^m_j \leq n^m_i(t) \tag{1}$$

where C^c_j and C^m_j identify the requested resources by the container. Similarly, suppose $V = \{v_1, v_2, \ldots, v_M\}$ represent the cluster of VMs in the IaaS cluster. Any request to be executed in a VM needs to meet its resource availability. Going by the above notation, if R_j is the j^{th} request of an application and $1 \leq l \leq M$,

$$R^c_j \leq v^c_l(t), \quad R^m_j \leq v^m_l(t) \tag{2}$$

A key metric that we target to improve in this work is application performance in terms of execution time latency. In order to not allow request processing time variations in different applications to hinder the overall performance tracking, we consider a relative response time parameter of a request when measuring performance. Relative Request Response Time (RRRT) is defined as the ratio between the standard (R^{r0}_j) and the actual response time (R^r_j) of a request. Standard response time is the time to response when the request is run in an isolated environment on a readily available resource. Accordingly, over the course of an application request workload, we target to minimize the average Relative Request Response Time (RRRT) ratio, i.e.,

$$Minimize : Average\ RRRT = \frac{1}{B}\sum_{j=1}^{B}\frac{R^r_j}{R^{r0}_j} \tag{3}$$

In addition to the performance goal, an equally important parameters for end users in any cloud service offering is the cost. Thus optimizing the overall infrastructure cost of running an application workload is our second key target. The cloud service provider cost model is different under a serverless and a serverful deployment model.

Existing commercial serverless platforms charge users based on the memory allocated (in MB) to the function instance (C^m_j), the execution time (in ms) of a request (E_j), and the number of requests received by the application (B). Thus if the charge per MB for 1ms is W, and the charge per request is L, the total cost of executing an application workload on the serverless platform is,

$$Cost_s = \sum_{j=1}^{B}\left(C^m_j \times E_j \times W\right) + L \tag{4}$$

An IaaS platform on the other hand charges users for the whole period that the infrastructure is rented out. If t_l is the time period (in seconds) that v_l was rented and p_l is the price per second for the same,

$$Cost_{VM} = \sum_{i=1}^{M} p_l \times t_l \tag{5}$$

Thus the infrastructure cost optimization objective could be stated as below:

$$Minimize : Cost_{Total} = Cost_s + Cost_{VM} \tag{6}$$

Accordingly our overall target objective is summarized below:

$$Minimize : Average\ RRRT + Cost_{Total} \tag{7}$$

IV. Deep Reinforcement Learning Model

A. Learning Model for Hybrid Scheduling

RL is a form of machine learning which predominantly works by learning through experience gathered by actively interacting with the problem environment. Based on the preferred outcome, the learning agent is rewarded whenever a 'better' action is taken. With enough experience, the agent ultimately learns to take actions leading to maximizing the cumulative reward along experience trajectories.

In this work, the RL agent is tasked with traversing a serverless and VM based hybrid computing environment, in order to determine a request scheduling policy for application workloads. Each time step of the agent corresponds to the event of receiving a user request at the hybrid load balancer discussed under the system model. The policy to be developed is aimed at achieving our objectives of time and cost. The key elements of the RL model are discussed below.

State space: The state space captures the important metrics in both the serverless and VM based environments in addition to the workload characteristics. Accordingly, the first part of the state vector carries the serverless cluster node specifications: [n^c_i, n^m_i, n^{idle}_i], which identify the free cpu, memory capacities and the number of idle (warm) containers in each node. The second part includes the request details: [R^c_j, R^m_j, R^{rate}_j, R^d_j], representing the requested cpu, memory, moving average arrival rate and the deployment mode of the previous request, respectively. The moving average rate of arrival is calculated over a time frame

which captures the total of the set up and the set idle time of a VM in the serverful cluster, which helps the model to gain an understanding on the historical load level as well as its duration. The mode of deployment in the preceding step gives an indication of the availability of warm/active instances, and thus is helpful in decision making. The final portion of the state vector is composed of details of the IaaS cluster: [v_l^c, v_l^m, v_l^s, v_l^t], referring to cpu, memory capacities, VM live status and the request waiting time for scheduling if selected. The last metric identifies how long it takes for the VM to come to 'ready' status and is calculated by the average time taken for a VM to start up and the remaining time since initializing the 'start' process.

Action space: Our action space takes a hierarchical form with two levels of decision making. The first level determines the deployment environment for the request while level two specifies the node of execution within the selected cluster. Accordingly, a complete action A can be represented as below:

$$A = [a_1, a_2] \begin{cases} a_1 \in \{serverless, serverful\} \\ a_2 \in \{n_1, n_2, \ldots, n_Q\}/\{v_1, v_2, \ldots, v_M\} \end{cases} \tag{8}$$

Compared to a combinatorial action space which does not distinguish between the two deployment modes, the RL agent is able to explore and learn the behavior of the hybrid environment faster with this formulation.

Reward: The step reward awarded to the agent after each action is aligned with the performance and cost objectives. We device two reward elements as below for action A_t:

R_1*:* The waiting time for the request to be scheduled. This is the resource creation time relevant as per the selected combined action. For a request directed to a node in the serverless platform, this would be equal to zero if there is any ready instance or one if a new container is to be created. If the VM cluster is selected for execution, this would either be zero or one for active or stopped VMs, else the remaining time for initializing as a fraction of the total, for a VM pending creation.

R_2*:* An approximation of the effective cost of running the request in the selected infrastructure. In the serverless cluster, this is the charge per request calculated as per (4). For the VM based cluster, we use an approximate value calculated as the cost of keeping the selected VM active during the execution time of the request, multiplied by the percentage of free resources in the VM at the time.

The total reward is the aggregate of both R_1 and R_2 above. Since we need to minimize the cumulative of these rewards in order to reach our targets, we insert a negative sign for this reward value when training the agent. Further, since R_1 and R_1 are in two different scales, we normalize them at each step.

B. Actor-Critic Based Hierarchical Scheduling Framework

Actor-critic methods in reinforcement learning make use of both the basic techniques of value based and policy based methods of finding the optimal policy for a given problem. Its fundamental architecture is designed with the use of two neural networks, the actor and the critic network. The actor is a policy network which uses an optimization method to train the network in the direction of the desired policy. Critic is driven by a value network which evaluates the policy generated by the actor.

Traditionally, actor-critic algorithms are implemented with one actor and one critic network. For our proposed hierarchical action space described above in Section IV-A, we design a network architecture with two actor networks and one critic network adapting the hybrid actor-critic architecture presented in [23] for parameterized action spaces. The two parallel actors work together to generate a complete action. The first actor performs the first level of action selection by learning a stochastic policy π_{θ_1}, while the second actor learns a policy π_{θ_2} in order to select the second action.

The policy optimization in the actor networks could utilize any policy gradient algorithm which works with discrete environments and suits the basic actor-critic architecture, such as Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO). Since TRPO is computationally intensive, we choose PPO for policy optimization in both the actor networks. It is proven to be an improved version of TRPO in terms of generalizability and its simplicity. PPO learns a stochastic policy π_θ by including a clipping function in its objective function and minimizing it.

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t] \tag{9}$$

where $r_t(\theta)$ identifies the probability ratio between the old policy and the new policy while ϵ is a hyper-parameter used to clip the objective function. The clipped function in PPO helps to keep the divergence of the old policy and the new policy within the trust region without having to use a constraint like in TRPO. In our proposed architecture, the two discrete policies π_{θ_1} and π_{θ_2} are updated separately by minimizing their respective clipped objectives during training.

The single critic network estimates the state-value function, $v_\pi(s_t|\phi)$ and learns by minimizing the difference between the target $(r + \gamma V_\phi(s'_t))$ and predicted values of the state, also known as the advantage function $A(s, a)$ as shown below.

$$\begin{aligned} J(\phi) &= r + \gamma V_\phi(s'_t) - V_\phi(s_t) \\ \phi &= \phi - \alpha \nabla_\phi J(\phi) \end{aligned} \tag{10}$$

where ϕ is the critic network parameter and $\nabla_\phi J(\phi)$ is the gradient of the network which is updated using gradient descent.

Algorithm 1 illustrates the pseudocode of the learning process of the proposed actor-critic based hierarchical scheduling framework. First we initialize the actor networks and the critic network with random weights and set the hyperparameters for model training (lines 1-2). At the start of each scheduling episode, the environment is reset. At each time step, the agent retrieves the state of the environment and feeds it to the first and second actor networks. The first actor maps the request to either the serverless or VM based environment, after which the second actor determines a scheduling node in that environment. Once the request is forwarded for execution, the agent receives a reward and the environment transitions to the next state. The transition data are stored in a memory buffer (lines 5-9). When an episode comes to an end, we train the networks S times by sampling a batch of step data from the memory, computing

Algorithm 1: Actor-Critic Based Hierarchical Scheduling Algorithm.

1: Initialize the two actor network and critic network parameters θ_1, θ_2 and ϕ
2: Initialize the training parameters α, β, and γ
3: **for** episode = 1 to E **do**
4: Reset the environment
5: **for** step = 1 to T **do**
6: Input the state s of the environment to actor networks $\pi_{\theta_1}(a|s)$ and $\pi_{\theta_2}(a|s)$
7: Select action a_1 and a_2 using using the first and second actor networks
8: Execute the combined action $A = (a_1, a_2)$, move to the next state s' and observe the reward r
9: Store the transition (s, a, r, s') in memory D
10: **for** j = 1 to S **do**
11: Randomly sample a mini-batch of samples of size K from memory D
12: **for** sample i = 1 to K **do**
13: Compute the loss and the gradients of the loss of the two actor $\nabla_{\theta_1} J(\theta_1)$, $\nabla_{\theta_2} J(\theta_2)$ and critic $\nabla_{\phi} J(\phi)$ networks
14: Update actor and critic network parameters θ_1, θ_2 and ϕ
15: Clear memory D
return

the network losses for each step and by updating the network parameters (lines 10-14). Finally the memory is cleared before the start of the next episode.

V. Performance Evaluation

A. RL Environment Design and Implementation

We build a serverless testbed with the Kubeless [24] open source serverless framework deployed on a Kubernetes [25] cluster, set up on the Melbourne Research Cloud [26]. This prototype environment is used to do initial resource profiling for the applications used in our experiments in addition to gathering various system behavioral parameters such as container creation (cold start) times etc.

Following the system architecture in our serverless prototype described above and the overall system model presented under Section III-A, we have developed a simulation environment for serverless and serverful hybrid scheduling of applications. The serverful/VM based functionalities in the simulator are based on Amazon EC2 VM instances. Further, this event-based simulator written in python is integrated with Keras [27] and Tensorflow(TF) [28] libraries in order to support our DRL model training. Although in this work we explore only scheduling techniques, our simulator is capable of evaluating novel RL-based solutions for many resource management related tasks including resource provisioning, scheduling, and scaling etc. As mentioned above, it supports applications deployed in a serverless, VM based or a hybrid environment and the source code is publicly available as an open-source software, 'Hybrid_DRL'.[1]

TABLE III
VM-Based Cluster Resource Details

Instance Type	vCPU cores	Memory(GB)	Quantity	Price($/hr)
m6a.large	2	8	5	0.108
t4g.xlarge	4	16	5	0.1696
m5.2xlarge	8	32	5	0.48
m5a.4xlarge	16	64	5	0.864

B. Experimental Settings

1) Cluster Setup: Our experiments are designed to include 20 nodes in the simulated serverless cluster. The processing power of each of these node vCPUs is considered to follow the clock speeds of the AWS Lambda invokers identified in [3], with 4 different vCPU count and memory configurations. The simulated serverful cluster is also composed of 20 VMs and their configurations are derived from the Amazon EC2 VMs (in Australia) closely matching the clock speed, vCPU and RAM configurations of the serverless nodes. The pricing model of these VMs are set as per the instance pricing model of the EC2 VMs, while AWS Lambda GB-second and per request rates are utilized for the serverless cluster cost calculations. The VM-based cluster resource details are summarized in Table III.

2) Workload Specifications: Serverless Applications: We select 10 applications from ServiBench [29] and FunctionBench [30] benchmark suites which are formed of a single function. These applications were chosen so that they have varying execution times which determine their sensitivity to cold start latencies. Further, each of them have different resource requirements and thus provides a diverse learning experience to the DRL agent, specially in terms of managing idling resources in the VM cluster. Our practical testbed is initially utilized for deriving resource metrics for the selected applications. A series of requests is sent to a ready instance of each application deployed in isolation, using the JMeter [31] load generation tool. The results from these tests collected by monitoring tools and averaged over multiple iterations are used to determine the resource consumption of a single function request (R_j^c, R_j^m) and its standard response time (R_j^{r0}). Table IV summarizes the nature of the selected benchmark applications.

Workload Creation: We utilize metrics from function traces exposed by Azure Functions [32] for a set of single function applications, when creating the training workloads. The per hour arrival rates for a particular function are extracted as the request rates, and a poisson distribution is followed when determining the inter arrival times of requests. Each workload is created with a single application receiving requests at fluctuating arrival rates, with each rate prevailing for different time durations. The DRL agent is trained with workloads of multiple applications with high and low request rates lasting for both short and long durations. In all the experiments, we maintain request arrival rates at 5-60 requests per second.

[1]https://github.com/Cloudslab/Hybrid_DRL

TABLE IV
APPLICATION DETAILS

Name	Resource Sensitivity	
	CPU	Memory
Primary	High	High
Float	High	High
Matrix Multiplication	High	High
Linpack	High	High
Load	low	low
Dd	High	Medium
Gzip-compression	High	Medium
Thumbnail Generator	Low	Medium
Image Processing	Medium	Medium
Video Processing	High	High

TABLE V
HYPER-PARAMETERS USED FOR DRL MODEL TRAINING

Parameter	Value
General	
Discount factor (γ)	0.99
Mini-batch size (K)	128
No. of training iterations per episode (S)	50
Optimizer	Adam
First Actor network parameters	
Learning rate (α_1)	1.00E-06
No. of input layers	1
No. of output layers	1
No. of hidden layers	2
No. of neurons in each hidden layer	150
Second Actor network parameters	
Learning rate (α_2)	1.00E-05
No. of input layers	2
No. of output layers	2
No. of hidden layers	4
No. of neurons in each hidden layer	150
Critic network parameters	
Learning rate (β)	5.00E-06
No. of input layers	1
No. of output layers	1
No. of hidden layers	2
No. of neurons in each hidden layer	150

3) Hyper-Parameter Configurations: Neural network training parameters for both the actor networks and also the critic network are decided on a trial and error basis. The discount factor is maintained at a high value since the scheduling decisions made over a period of time affect the success of the agent's decision making in terms of both the mode of deployment as well as the execution nodes. The second actor initially has a higher learning rate, so that the agent learns to take better node selection decisions during initial iterations, without which the first level decision of environment selection too will have no value. A decay factor is used to gradually bring down this learning rate to match that of the first actor subsequently. The critic constantly maintains a relatively higher discount rate since in the actor-critic architecture, the actors largely rely on the feedback and guidance of the critic network. These neural network parameters, and the other settings for training the DRL agent are listed in Table V.

C. Performance Metrics

We use two metrics to evaluate the effectiveness of our solution noted below:

1) *Average Relative Request Response Time (RRRT):* The average, relative request response time of an application workload during an episode, calculated using (3).
2) *User Cost:* The total cost of running an application workload in the hybrid cluster environment. The calculation of this metric is as in (6).

D. Baseline Scaling Techniques

We use three baseline techniques to compare the performance of our proposed solution.

VM-only: The entire workload execution takes place on a VM-based cluster.

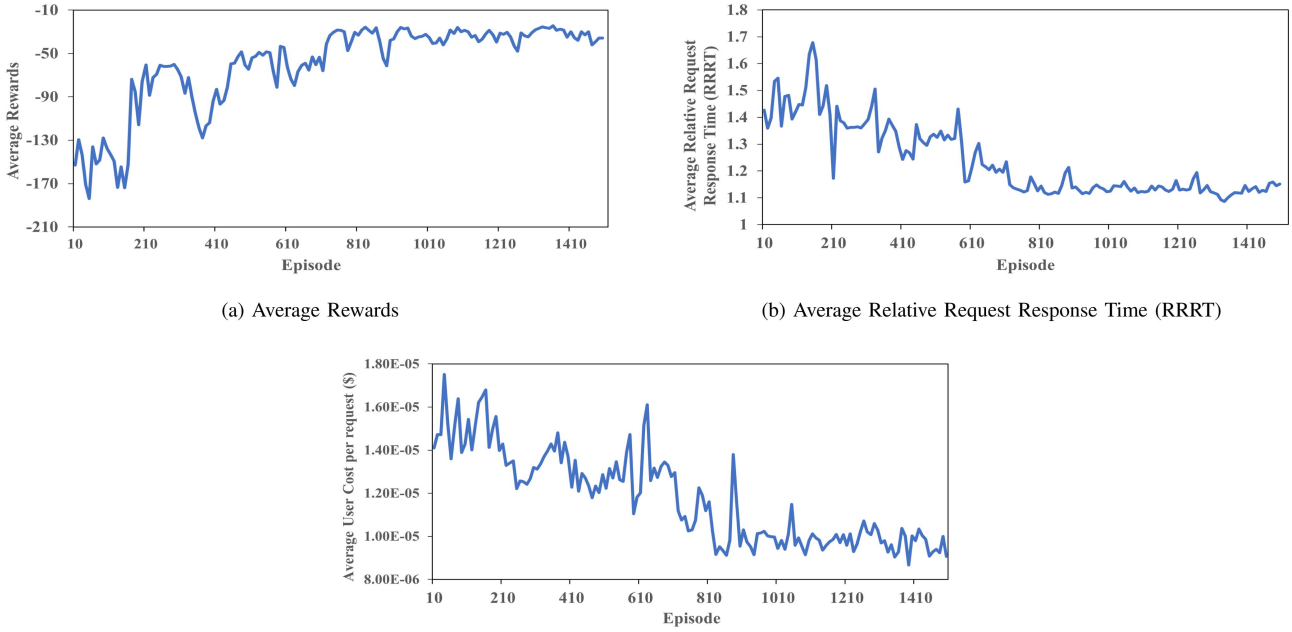
S-Only! The entire workload execution takes place on a serverless cluster.

Std-A2C: DRL model trained with the standard actor-critic network architecture with a single actor network. The two levels of decision making are accommodated by composing a combinatorial action space where each action has two decision elements.

E. Convergence of the DRL Model

The graphs in Fig. 2 illustrate the step-by-step training progress achieved by the hierarchical actor-critic agent, H-A2C across iterations. We demonstrate the progress in terms of the cumulative reward over an episode, the average RRRT experienced and the average user cost incurred for executing a request during an episode. Note that each marked value in the graphs corresponds to the average over 10 episodes for better readability and ease of understanding.

It is clear that the model reaches convergence around the 900^{th} iteration, after which the achieved progress is maintained. Fig. 2(a) shows how the episodic reward gradually improves and reaches convergence, as the agent learns a policy that is able to take decisions that optimize the constituents of the reward metric at each scheduling step. Fig. 2(b) and (c) illustrate the gradual reduction in overall RRRT and the incurred cost for requests with convergence, respectively. The logic behind this achievement is two-fold, since the agent has a hierarchical decision structure. The first actor network learns to select a deployment environment that leads to lower relative response times by targeting lesser frequency in cold starts, considering the nature of the application specially in terms of its execution time. The cold starts refer to the container creation time in a serverless execution as well as the time for setting up new resources in a VM setting if all active resources are exhausted. While aiming for a reduced RRRT, the model also learns to aim for an environment where the marginal cost for each request would be minimal. This could be based on the availability of warm function instances in a serverless setting or active VMs with free resources in a VM

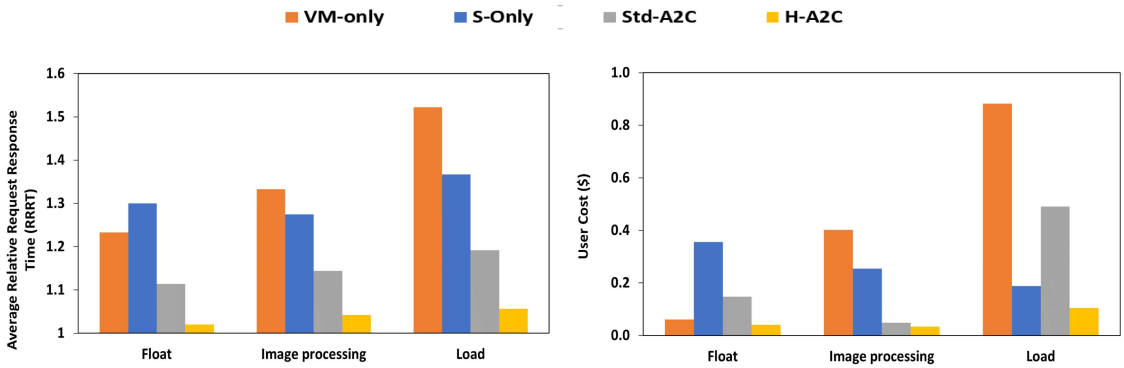


(a) Average Rewards

(b) Average Relative Request Response Time (RRRT)

(c) Average User Cost per request ($)

Fig. 2. Training progress of the DRL agent in terms of the agent rewards, average RRRT, and the average user cost per request.



(a) Average Relative Request Response Time (RRRT)

(b) Total User Cost ($)

Fig. 3. Comparison of the average RRRT and the total user cost incurred by different application workloads, achieved by the H-A2C model and the baseline algorithms.

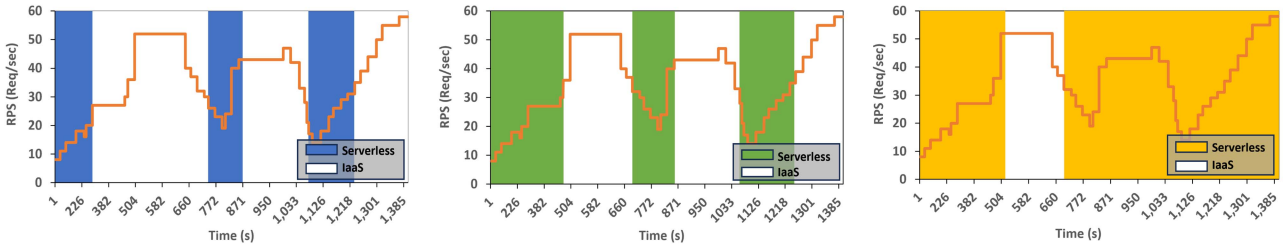
setting. Subsequent to the first actor's decision, the second actor uses its learned policy to select an execution node in the selected environment, that further elevates the target metrics. A node with ready instances could be the better choice in a serverless setting, while an active VM with higher utilization levels which leads to lesser idling times, could be selected in a VM-based setting.

F. Analysis of Model Performance on the Evaluation Data Sets

The trained model is evaluated in terms of the response time and user cost performance achieved over the evaluation workloads. The workloads for these experiments are created by following trace snippets from Wikipedia [33] to simulate request arrival times. The Fig. 3(a) and (b) show the results averaged over five different workloads, run for each of the applications float, load and image processing, which have been selected out of the ten applications used in the experiments, due to space limitations. These three applications were specifically chosen for demonstration of model performance due to their distinction in resource consumption and execution times. Further, Figs. 4 and 5 illustrate the behavior of the agent decision model for two of the evaluation workloads, with the points of switching deployments between the serverless and IaaS clusters.

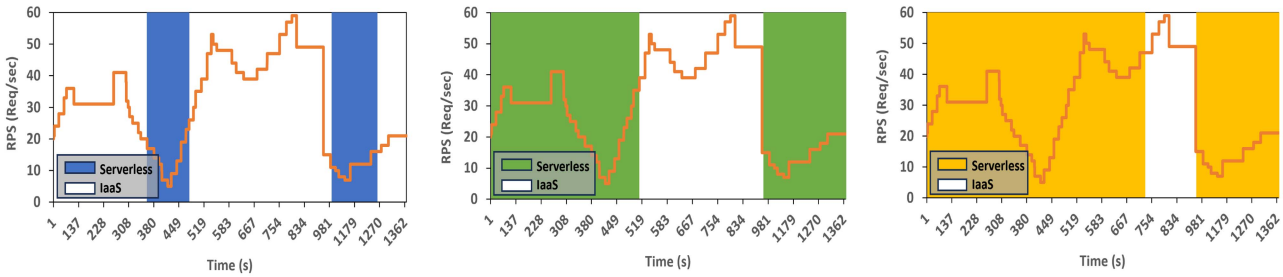
1) Evaluation of Application Performance: Application performance is evaluated in terms of the achieved average RRRT (Fig. 3(a)) for each application by following the trained policy of our hierarchical A2C (H-A2C) model and the other baseline algorithms, for scheduling workload requests.

Our H-A2C model is able to demonstrate the best performance for all three applications, with overall relative response time improvements of up to 11%. This is achieved by carefully directing each request to the execution environment that is more likely to



(a) Float (b) Image Processing (c) Load

Fig. 4. Workload-1: Deployment switch between Serverless and IaaS clusters for the three applications.



(a) Float (b) Image Processing (c) Load

Fig. 5. Workload-2: Deployment switch between Serverless and IaaS clusters for the three applications.

have ready resources for the upcoming request traffic as shown in Figs. 4 and 5. Once the deployment mode is decided, the trained agent is also able to choose the best out of the available cluster nodes. Both these decisions are taken to suit each phase of the workload considering the prevailing traffic patterns. The trained S-A2C model shows the next best performance, but fails to capture the fine details of each cluster environment, that is enabled by the hierarchical nature of the H-A2C model.

The float application has the highest CPU and memory consumption and the longest execution time out of the three. Thus when the requests are scheduled solely on a VM cluster, it is able to maintain a relatively high utilization level in the rented resources even at low load levels, in the long term. This triggers auto-scaling of VM resources in the cluster, leading to lesser request latency effects arising from new VM initialization. However, when the entire workload is executed as serverless functions, due to the high application response time, the availability of warm idling containers is often limited. Thus the s-only execution shows the worst performance resulting from frequent cold start of instances (first graph of Fig. 3(a)). To overcome the shortcomings of both these scenarios, our H-A2C model resorts to serverless deployments during periods of very low and fluctuating traffic levels, and switches to the VM cluster as the load starts to increase and then stabilizes, as seen in the graphs Figs. 4(a) and 5(a).

In contrast, the load application results in the need for frequent startup of new VMs in the IaaS cluster to accommodate new requests. This is due to its very low execution time and resource requirements. For the same reasons, the effect of these added latencies on the relative response time too is high for the load application. This situation escalates during periods of low and irregular traffic patterns, when VMs are switched on and off often due to idling. On the other hand, the s-only execution performance is not much different from the float application scenario since the increase in latency is mostly only due to the higher relative effect of the cold start delays, arising from the low application response time. In comparison to these two strategies, we can see that the H-A2c model leverages both these clusters by executing the application requests as functions for the majority of the workload, while utilizing the VM cluster only during periods of lasting high traffic levels (Figs. 4(c) and 5(c)). During these traffic bursts, the load level seems to be sufficient to maintain a set of ready VMs, thus compensating for the cold start latencies that the serverless execution would otherwise entail.

The image processing application possesses a median response time and resource needs. Thus its relative latency effects are less evident and more prominent compared to that of the load and float applications respectively.

2) Evaluation of Incurred User Cost: The cost charged to the user is measured as the total billed amount for the rented VM resources and function executions, by each application over the duration of the workload. Fig. 3(b) shows that the scheduling decisions made using the H-A2C model result in the least cost charged to the user for all three applications, with the load application attaining a cost reduction of approximately 44%. The trained policy is equipped with knowledge on the workload and system behavioural patterns, so as to make informed decisions on when to switch deployments to each cluster environment, that are cost efficient. The reduction in VM idling time and maximizing the usage of the serverful cluster by maintaining just the right amount of resources active, is the key enabler for overall user cost minimization. Since the per request charge for a serverless function execution is quite high, the scheduler chooses to use it only during periods of fluctuating and low load levels, during

which the use of rented VMs results in cost inefficiencies. The lowest overall cost under the H-A2C model is incurred by the image processing application due to its better use of both the environments compared to the load application, in addition to the lower actual resource consumption than the float application. The Std-A2c model too follows this cost pattern.

The VM-only deployment results in worst performance for the load application due to high VM idling times, with the highest incurred cost for the workload execution out of all the applications. The S-only execution cost difference for the three applications is only dependent on the consumed level of resources and the workload execution times, as per the serverless billing model.

VI. Conclusions and Future Work

With the increased adoption of the serverless cloud model for different application domains, studies have shown that limitations also exist in these environments that hinder the achievement of the best possible performance for certain workloads.

In this paper, we presented a DRL based hybrid execution model for application workloads, which utilizes both a serverless and a serverful cluster environment. The proposed scheduling framework involves a hierarchical decision model, where the DRL agent first learns to choose the best mode of execution for an application request. Thereafter, it proceeds to decide the node that is most suitable for the request execution within the selected cluster environment. The DRL agent follows an actor-critic architecture and the reward model is set targeting relative application latency and the resource cost charged to the user as the optimization objectives. The model evaluation experiments show that the users are able to avoid application latencies arising from frequent container cold starts in a serverless environment, as well as the problem of over/under utilization of rented infrastructure in a VM setting, by adopting a carefully designed hybrid system architecture.

As part of future work, we plan to expand our proposed hybrid scheduling framework to support multi-function applications with Directed Acyclic Graph (DAG) based workflow structures. In such a scenario, each request would contain multiple subtasks, each of which would require hierarchical decision making for environmental and node selection for its deployment. Each sub-task would have dependency relationships with each other, which need significant attention in the decision making process. In addition, the treatment for data transfer latencies among tasks would need to be carefully investigated depending on the deployment cluster. The DRL agent holds the responsibility of learning these complex task dependencies as well as input and output data requirements for each request component.

References

[1] A. Mampage, S. Karunasekera, and R. Buyya, "A holistic view on resource management in serverless computing environments: Taxonomy and future directions," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 1–36, 2022.

[2] J. R. Gunasekaran, P. Thinakaran, M. T. Kandemir, B. Urgaonkar, G. Kesidis, and C. Das, "Spock: Exploiting serverless functions for SLO and cost aware resource procurement in public cloud," in *Proc. IEEE 12th Int. Conf. Cloud Comput.*, 2019, pp. 199–208.

[3] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *Proc. 2018 USENIX Annu. Tech. Conf.*, 2018, pp. 133–146.

[4] J. H. Novak, S. K. Kasera, and R. Stutsman, "Cloud functions for fast and robust resource auto-scaling," in *Proc. IEEE 11th Int. Conf. Commun. Syst. Netw.*, 2019, pp. 133–140.

[5] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, and A. Gandhi, "Ensure: Efficient scheduling and autonomous resource management in serverless environments," in *Proc. IEEE Int. Conf. Autonomic Comput. Self-Organizing Syst.*, 2020, pp. 1–10.

[6] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, "Mashup: Making serverless computing useful for hpc workflows via hybrid execution," in *Proc. 27th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2022, pp. 46–60.

[7] Z. Li et al., "Amoeba: QoS-awareness and reduced resource usage of microservices with serverless computing," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2020, pp. 399–408.

[8] S. Horovitz, R. Amos, O. Baruch, T. Cohen, T. Oyar, and A. Deri, "Faastest-machine learning based cost and performance faas optimization," in *Proc. 15th Int. Conf. Econ. Grids Clouds Syst. Serv.*, Pisa, Italy, Springer, Sep. 18–20, 2019, pp. 171–186.

[9] A. Reuter, T. Back, and V. Andrikopoulos, "Cost efficiency under mixed serverless and serverful deployments," in *Proc. IEEE 46th Euromicro Conf. Softw. Eng. Adv. Appl.*, 2020, pp. 242–245.

[10] A. Mampage, S. Karunasekera, and R. Buyya, "Deep reinforcement learning for application scheduling in resource-constrained, multi-tenant serverless computing environments," *Future Gener. Comput. Syst.*, vol. 143, pp. 277–292, 2023.

[11] H. Yu, A. A. Irissappane, H. Wang, and W. J. Lloyd, "FaaSRank: Learning to schedule functions in serverless platforms," in *Proc. IEEE Int. Conf. Autonomic Comput. Self-Organizing Syst.*, 2021, pp. 31–40.

[12] C. K. Dehury, S. Poojara, and S. N. Srirama, "Def-DRel: Systematic deployment of serverless functions in fog and cloud environments using deep reinforcement learning," 2021, *arXiv:2110.15702*.

[13] X. Yao, N. Chen, X. Yuan, and P. Ou, "Performance optimization of serverless edge computing function offloading based on deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 139, pp. 74–86, 2023.

[14] Q. Tang et al., "Distributed task scheduling in serverless edge computing networks for the internet of things: A learning approach," *IEEE Internet Things J.*, vol. 9, no. 20, pp. 19634–19648, Oct. 2022.

[15] H. Jeon, S. Shin, C. Cho, and S. Yoon, "Deep reinforcement learning for QoS-aware package caching in serverless edge computing," in *Proc. 2021 IEEE Glob. Commun. Conf.*, 2021, pp. 1–6.

[16] G. Somma, C. Ayimba, P. Casari, S. P. Romano, and V. Mancuso, "When less is more: Core-restricted container provisioning for serverless computing," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2020, pp. 1153–1159.

[17] A. Zafeiropoulos, E. Fotopoulou, N. Filinis, and S. Papavassiliou, "Reinforcement learning-assisted autoscaling mechanisms for serverless computing platforms," *Simul. Modelling Pract. Theory*, vol. 116, 2022, Art. no. 102461.

[18] P. Vahidinia, B. Farahani, and F. S. Aliee, "Mitigating cold start problem in serverless computing: A reinforcement learning approach," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 3917–3927, Mar. 2023.

[19] H. Qiu et al., "Reinforcement learning for resource management in multi-tenant serverless platforms," in *Proc. 2nd Eur. Workshop Mach. Learn. Syst.*, 2022, pp. 20–28.

[20] Z. Zhang, T. Wang, A. Li, and W. Zhang, "Adaptive auto-scaling of delay-sensitive serverless services with reinforcement learning," in *Proc. 2022 IEEE 46th Annu. Comput. Softw. Appl. Conf.*, 2022, pp. 866–871.

[21] EC2 on-demand instance pricing – amazon web services, 2023. Accessed: Jul. 07, 2023. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/

[22] Instance auto scaling - Amazon EC2 autoscaling - AWS, 2023. Accessed: Jul. 07, 2023. [Online]. Available: https://aws.amazon.com/ec2/autoscaling/

[23] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid actor-critic reinforcement learning in parameterized action space," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 2279–2285.

[24] Kubeless, "Kubeless," 2021. Accessed: Jan. 13, 2022. [Online]. Available: https://kubeless.io/

[25] "Kubernetes," 2022. Accessed: Apr. 08, 2022. [Online]. Available: https://kubernetes.io/

[26] Melbourne research cloud documentation, 2022. Accessed: Jul. 22, 2022. [Online]. Available: https://docs.cloud.unimelb.edu.au/

[27] Keras, "Keras: The Python deep learning API," 2021. Accessed: Jan. 20, 2022. [Online]. Available: https://keras.io/
[28] TensorFlow, "Tensorflow," 2021. Accessed: Jan. 20, 2022. [Online]. Available: https://www.tensorflow.org/
[29] J. Scheuner et al., "Let's trace it: Fine-grained serverless benchmarking using synchronous and asynchronous orchestrated applications," 2022, *arXiv:2205.07696*.
[30] J. Kim and K. Lee, "Functionbench: A suite of workloads for serverless cloud function service," in *Proc. IEEE 12th Int. Conf. Cloud Comput.*, 2019, pp. 502–504.
[31] A. S. Foundation, "Apache jmeter - Apache jmeter™," 2021. Accessed: Nov. 08, 2021. [Online]. Available: https://jmeter.apache.org/
[32] M. Shahrad et al., "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proc. USENIX Annu. Tech. Conf.*, 2020, pp. 205–218.
[33] Wikipedia access traces | wikibench, 2020. Accessed: Dec. 02, 2020. [Online]. Available: http://www.wikibench.eu/?page_id=60

Anupama Mampage received the BSc engineering (Hons) degree, specialized in electronic and telecommunication engineering from the University of Moratuwa, Sri Lanka, in 2017. She is currently working toward the PhD degree with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia. Her research interests include serverless computing, internet of things (iot), distributed systems and reinforcement learning.

Shanika Karunasekera (Member, IEEE) received the BSc degree in electronic and telecommunications engineering from the University of Moratuwa, Sri Lanka, in 1990 and the PhD degree in electrical engineering from the University of Cambridge, U.K., in 1995. She is currently a professor with the School of Computing and Information Systems and the deputy dean (Academic) with the Faculty of Engineeering and IT, University of Melbourne, Australia. Her research interests include distributed computing, mobile computing, and social media analytics.

Rajkumar Buyya (Fellow, IEEE) is a Redmond Barry distinguished professor and director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory with the University of Melbourne, Australia. He has authored more than 625 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=160, g-index=345, 137100+ citations).