ImpalaE: Towards an optimal policy for efficient resource management at the edge

Tania Lorido-Botran[1], *Muhammad Khurram* Bhatti[2]

[1]*Roblox, Bilbao, Spain*

[2]*Information Technology University, Arfa Software Technology Park, Ferozepur Road, Lahore, Pakistan*

Abstract. Edge computing is an extension of cloud computing where physical servers are deployed closer to the users in order to reduce latency. Edge data centers face the challenge of serving a continuously increasing number of applications with a reduced capacity compared to traditional data center. This paper introduces *ImpalaE*, an agent based on Deep Reinforcement Learning that aims at optimizing the resource usage in edge data centers. First, it proposes modeling the problem as a Markov Decision Process, with two optimization objectives: reducing the number of physical servers used and maximize number of applications placed in the data center. Second, it introduces an agent based on Proximal Policy Optimization, for finding the optimal consolidation policy, and an asynchronous architecture with multiple workers-shared learner that enables for faster convergence, even with reduced amount of data. We show the potential in a simulated edge data center scenario with different VM sizes based on Microsoft Azure real traces, considering CPU, memory, disk and network requirements. Experiments show that *ImpalaE* effectively increases the number of VMs that can be placed per episode and that it quickly converges to an optimal policy.[1].

Keywords: edge computing, policy gradient, reinforcement learning, efficient resource management

1. Introduction

Cloud Computing providers have popularized and quickly replaced private data centers. Many businesses, government organizations and research centers rely on external clouds to run their workloads. However, Cloud data centers are usually located far away from the end-user and the perceived latency might not be up to the standard. In recent years, the Edge Computing paradigm has augmented Cloud capabilities by placing computing facilities and services close to end users. Thus, Edge data centers are able to provide low latency and mobility to delay-sensitive applications. According to a Markov Growth study [10], Edge Computing was valued at USD 1.93 Billion in 2018 and is projected to reach USD 10.96 Billion by 2026. With this high growth in revenue, it is clear the increased interest in this services.

The Edge computing platform is expected to deliver consistent performance despite the rapid increase of application demand, specially coming from Internet-of-Things applications, such us self-sufficient vehicles producing data from their various cameras, radar or accelemerometers. The new challenge for edge service providers is to perform efficient resource management of their edge data centers with reduced computation and storage capabilities [6]. In particular, providers will look for automated solutions that can adapt to the varying demand and diverse workloads.

Reinforcement Learning (RL) is a family of self-adaptive algorithms that has been successfully applied to multiple domains. From the popular AlphaGo [16] for playing the game of Go, to autonomous driving, drug discovery, personalized recommendations and optimizing chemical reactions. RL has also been applied to for cloud resource optimization, both horizontal and

vertical scalability [2, 19, 20]. Similarly, RL has the potential to provide and efficient and automated solution to the management of resource at the Edge.

2. Related work

Edge computing has received increasing attention in recent years. A common use case scenario is the off-loading of certain requests to different Edge data centers. Liu et al. [8] focus on the task scheduling problem and proposed an RL-based scheduling solution and successfully offioad certain tasks to other data centers. Some authors have proposed DRL-based solutions for the offioading of VMs [14]. However, computation offioading might lead to unbalancing issues, as some edge data centers in the region could be overloaded while some others are in idle state [1]. Unbalanced data centers lead to performance degradation and wasted resources. One approach would be to spread the load equally among the difference edge data centers. Puthal et al. [13] take this approach and propose a solution based on Bread-First-Search to keep the application load equally distributed. However, edge data centers are characterized from scarce resources compared to traditional servers and a load balancing approach will not maximize the number of applications that can be served.

There are clashing objectives between the end-user and the service provider. The end-user expects guaranteed application performance, while the provider wants to maximize its revenue by increasing the number serviced applications. In order to meet both end-user and provider's expectation, it seems reasonable to define the overall objective as a consolidation problem: placing as many requests as possible using the minimal capacity, always subject to resource constraints. With this goal in mind, some authors have focused on the execution of tasks on edge data centers [8, 21]. Zhu et al. [21] successfully introduce two approximation scheduling algorithms focused on minimizing energy consumption and reducing the overall task execution delay.

As stated by Khan et al. [6], edge data centers can benefit from the use of Virtual Machines to co-allocate multiple applications in the same physical server. Tao et al. [17] gather a list of proposed solutions that handle the VM placement on edge data centers. Proposed optimization methods range from Mixed-Linear Non-Linear Programming [4, 12] to Particle Swarm Optimization [7]. However, there seems to be a lack of solutions exploring the potential of RL for optimal VM placement in edge data centers, aiming at minimizing resource wastage.

To the best of our knowledge, this is the first attempt to explore the application of policy-gradient RL methods to achieve efficient resource management in edge data centers. This paper introduces an agent (named *ImpalaE* that uses policy-gradient method to find the optimal placement policy and a distributed architecture that enables fast training. The resource management

problem is formulated with a bi-objective function that tries to (1) reduce the number of physical servers utilized and (2) maximize the number of applications that can be placed in the edge data center.

3. Background: Policy-Based Reinforcement Learning

The basic elements in an RL problem are the *agent* and the *environment*. The agent continuously interacts with the environment, observes the current *state* and decides the best *action* to take. After some time, the agent will observe the *reward* obtained after applying that action. The goal is to learn an optimal *policy* $\pi_\theta(a|s)$ that maps each state with its optimal action.

3.1. Vanilla Policy Gradient (PG)

There are different approaches to learn the optimal policy. As the name suggests, *Policy-based* algorithms directly learn the policy without an intermediary function. The policy $\pi_\theta(a|s)$ is approximated with deep neural network that has a vector of policy parameters θ. The goal is to adjust the values of these parameters, such that the policy maximizes the reward obtained from the environment.

Policy gradient methods rely on applying stochastic gradient descent as an iterative process. At each step, the algorithm estimates the gradient of some estimated scalar performance objective $J(\pi_\theta)$ and updates the policy parameters θ:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta) \tag{1}$$

The gradient of $J(\pi_\theta)$ for the *Vanilla Policy Gradient* can be calculated as follows:

$$\nabla_\theta J(\pi_\theta) = \mathop{E}_{\tau \sim \pi_\theta} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t), \tag{2}$$

where τ is an episode, that is a sequence of states and actions, e.g. a pre-defined sequence of requests and their corresponding placements in the edge data center; and E denotes calculating average over a batch of samples.

The main drawback in Vanilla PG is the high gradient variance, that will hinder the convergence to an optimal policy. The advantage function A^{π_θ} included in the gradient function helps in reducing such variance. Without going deep into the details, the advantage function evaluates how good an action is compared to the average action for a specific state.

3.2. Proximal Policy Optimization (PPO)

PPO [15] aims to optimize the gradient update taken at each step, ensuring that it minimizes the objective function, while ensuring that the difference to the previous policy is relatively small. Too big of an update might cause a divergence from the optimal policy. PPO imposes a constraint to the policy gradient updates as follows:

$$L(\theta) = L^{CLIP}(\theta) = E_t\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{3}$$

There are two main modifications with respect to the vanilla PG method. The first one is $\square_{\square} = \frac{\square_{\square}(\square_{\square}|\square_{\square})}{\square_{\square_{\square\square\square}}(\square_{\square}|\square_{\square})}$, which computes a ratio between the current policy (after update) and the older policy (just before the update). Additionally, PPO relies on a clipping function $\square\square\square\square(\square_{\square}(\square), 1 - \square, 1 + \square)$ that keep the value or $\square_{\square}$ between certain range defined by $1 - \square$ and $1 + \square$.

PPO with Clipping is used as the core agent for *ImpalaE*. The full logic is depicted in Algorithm 1:

Algorithm 1: PPO with clipping

Input: initial policy parameters $\square_0$, clipping threshold $\square$
for *0,1,2, . . . do* **do**
  Collect set of partial trajectories (episodes) $\square$ on policy $\square = \square(\square)$
  Estimate advantages $\square_{\square}$ using any advantage estimation algorithm Update the policy by maximizing the policy the PPO-Clip objective:
  $\square_{\square+1} = \arg\max_{\square} \square_{\square\square\square\square}(\square_{\square})$, typically, by taking $\square$ steps of minibatch stochastic gradient descent with Adam optimization
end

3.3. Importance Weighted Actor-Learner Architectures (IMPALA)

IMPALA [3] is a state-of-the-art algorithm produced by DeepMind. It uses the vanilla Policy Gradient at its core, but also introduces two significant improvements: a distributed architecture, and a correction algorithm V-trace. First, it introduces a highly-scalable architecture that relies on a single (or multiple) learner and multiple workers (see figure 1. In traditional RL approaches [11], each worker updates its local model parameters before each episode and communicates gradients to the main learner. IMPALA proposes a loosely coupled architecture where each worker focuses on collecting trajectories of experience (states, action, rewards). Then, the learner asynchronously samples batches of experiences from the workers, computes the policy gradients and updates the current model. This architecture enables the learner to be accelerated by a GPU and to distribute the workers across different nodes and collect experience on different domains (e.g. independent edge data centers).

The high scalability of the IMPALA architecture comes at a cost. Each worker interacts with its environment based on a policy that is slightly older than the main learner's policy, since the learner broadcasts the updated weights in a period and asynchronous manner. In order to address this divergence, Espeholt et al. [3] introduce a correction algorithm called *V-trace* that readjusts the value function $\square(\square)$ for each state and account for the lag in each action decision.

4. *ImpalaE*: efficient resource management at the Edge

This paper introduces *ImpalaE*, an agent designed to address the specific resource management needs from Edge Computing paradigm. The agent specializes in edge data centers that use Virtual Machines as an abstraction layer to place applications. It relies on the use of Policy Gradient Reinforcement Learning to learn and adapt to different VM request arrival patterns and dynamic resource usage. By leveraging a combination of PPO with an asynchronous architecture,

it quickly finds the optimal placement policy that squeezes the maximum performance out of the reduced capacity of an edge data center. As a first step, the Edge computing environment is formulated to be suitable for an RL-based agent.

4.1. Environment modeling

The scenario is one or more edge data centers composed of $\square$ physical servers. Each physical servers has a given capacity for a set of resources, $\square$. The agent has to learn the optimal policy $\square$ that matches each incoming request, expressed as a VM type with specific resource requirements, with the best physical server available. The overall goal is to maximize the number of requests that can be served given the current capacity. With this goal in mind, the resource management problem on edge data centers can be formulated as a Markov Decision Process (MDP) as follows:

State space: The state $\square$ at time $\square$ is defined as the current resource usage in the data center, together with the request received at time $\square$. The resource usage of each physical server is expressed as a normalized variable, ranged [0, 1], for each of the resources considered $\square$. Additionally, each physical server has a binary variable associated $\square$, which indicates if it is active (it has any load assigned to it) or not. Overall, the resource usage of the data center is a multi-dimensional vector $[\square, \square + 1]$. Each request $\square$ corresponds to a VM type, defined a set of $\square$ resource requirements that need to be satisfied. For the current case, we will consider $\square = 4$ resources, namely CPU, memory, disk and network capacity.

Action space: The action space $\square$ is the set of $\square$ physical servers available in the data center. At time $\square$, $\square_\square$ is defined as the subset of servers where the current request $\square$ could be placed, that is, never exceeding the capacity of the machine:

$$\square_\square = \{\square \in \square \mid \sum_{\square=1}^{\square} \square_{\square,\square} + \square_\square \leq 1\} \tag{4}$$

where $\square_{\square,\square}$ is the current utilization value for physical server $\square$ and resource $\square$ and $\square_\square$ is the capacity requested for resource $\square$.

Reward definition: The primary goal in the edge data center is to maximize the number of requests that can be served with the available capacity. The reward function $\square$ is defined with this goal in mind and it is composed of two objectives. The first objective $\square_1$ accounts for the amount of unused resources in the data center, normalized by the total capacity, $\square * \square$:

$$\square_1 = -\frac{\sum_{\square=1}^{\square} \square_\square * \square_\square}{\square * \square} \tag{5}$$

where $\square_\square$ is the total amount of free capacity across $\square$ resources for physical server $\square$. The reward only accounts for free resources in active physical servers, defined with $\square_\square = 1$.

The second part of the reward function directly accounts for the number of requests remaining to be placed in the current episode:

$$\square_2 = -\frac{\square - \square}{\square} \tag{6}$$

The final reward function is simply the linear combination of $\square_1$ and $\square_2$ with equal weights.

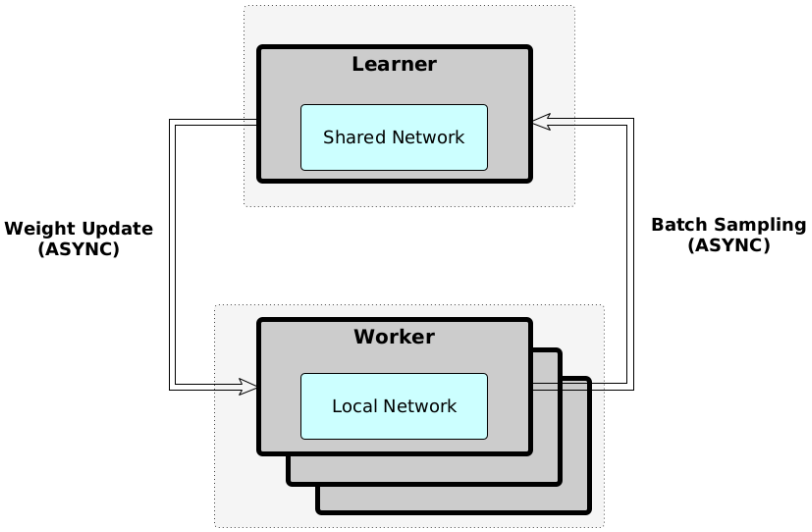

Figure 1: Architecture for *ImpalaE*.

4.2. Agent architecture

The proposed agent is based on the asynchronous architecture introduced by [3], from which it takes its name, *ImpalaE*. It consists of a main learner and one or more workers (see figure 1). Each worker interacts with the environment using their local copy of the network (only performing inference) and store (state, action, reward) samples. The main learner asynchronously samples batches from each of the workers and uses them to update the central network. After that, the learner broadcasts the network updated new weights to each of the learners in an asynchronous manner. This architecture enables for faster, parallel collection of environment info, which in turn leads for a quick convergence toward the optimal policy.

The learner is based on PPO algorithm with clipping (see Algorithm 1) for finding the optimal policy, that is, the best placement of each incoming VM request to the edge data center. The network model uses a shared architecture for the policy and the value function. It consists of feed-forward neural network with TanH activation function. In order to speed up the convergence, the learner makes use of a *buffer replay*. This buffer stores all the instances composed of (□□□□□, □□□□□□, □□□□□□, □□□□_□□□□□). Periodically, the learner samples □□□□*h*_□□□□ is sampled from the buffer to perform a gradient update in the policy network. Finally, the learner leverages *V-trace*[3], a correction algorithm that fixes discrepancies in the instances as a result of the asynchronous architecture. Table 1 contains a summary of the configuration used in the experimental evaluation:

5. Experimental evaluation

The following set of experiments are defined to evaluate the general performance of *ImpalaE*, compared against other policy-gradient methods from the state-of-art, and also the convergence

Table 1
Parameter configuration for *ImpalaE*.

Type	Parameter	Symbol	Value
Scenario	Number of physical servers	□	500
	Number of resources	□	4
	Number of actions	|□|	□
ImpalaE	Learning rate	□	0.005
	Train Batch size		500
	Optimization algorithm		Adam
	Clipping parameter		0.4
	Number of workers		2
Network Model	Input layer		(□ + 1)*(□ + 1)
	Hidden layer 1		1024
	Hidden layer 2		1024
	Output layer		□

and scalability of the agent architecture.

Testing environment: A simulated environment of an edge data center with certain number of homogeneous physical servers (same capacity). Each physical server and VM request is defined in terms of their CPU, memory, network and disk requirements. The resource specification is normalized between 0 and 1 (required by the model input). The experiments are based on real-world traces collected from Microsoft Azure data center [5, 18] (in particular, 15 VM types assigned to a machine identified with id 0). All algorithms are implemented in Python v3.8 and models are implemented using Tensorflow v2.5.0, and trained on a GPU. The hardware for the experiments is a machine with Intel Cor i7-10510U, 16GB of RAM, NVIDIA GeForce MX330.

Baseline methods: *ImpalaE* is compared against one heuristic method, Round Robin, and two other state-of-the-art RL algorithms: (vanilla) Policy Gradient (PG) and Proximal Policy Optimization (PPO).

5.1. Convergence and performance evaluation

The main goal of *ImpalaE* is to quickly converge to the optimal placement policy, the one that optimizes resource usage and maximises the number of requests that can be accommodated in the edge data center. In the first scenario, the data center is composed of 500 physical servers and has enough capacity to serve an episode consisting of 1000 VM requests. Requests are randomly drawn from a set of 14 VM types extracted from Azure data center traces (machineID 0). For fairness of results, the same network architecture is used for *ImpalaE*, PPO and PG. The network contains 2 hidden layers, with 1024 units each. When the agent architecture allows, two workers are used in the training process.

Figure 2 shows the convergence results for *ImpalaE*, PPO, PG and Round Robin. In less than 30 iterations, *ImpalaE* quickly converges to the optimal policy. In contrast, both PG and PPO achieve a sub-optimal policy (lower than the heuristic-based agent, Round Robin), with lower mean reward per episode. PG takes a high number of iterations to converge.

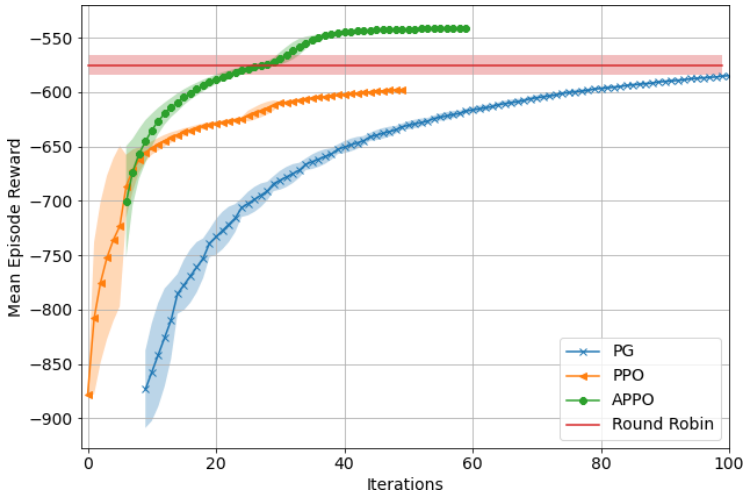

Figure 2: Training results for *ImpalaE*, PPO and PG.

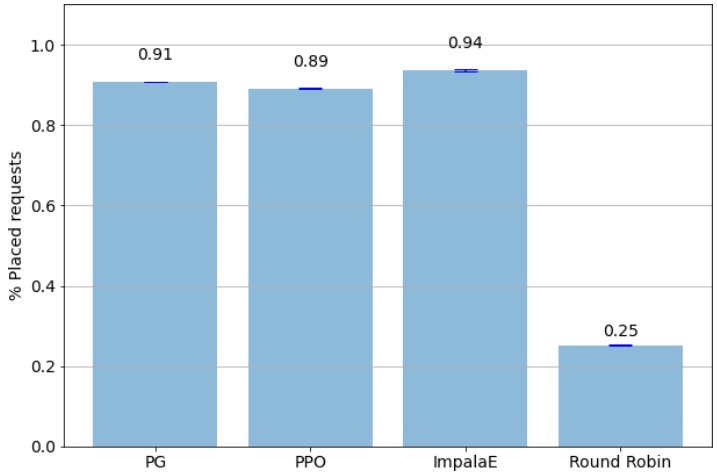

Figure 3: Mean percentage of placed requests per episode.

The second scenario is designed to stress the agent ability to make optimal placement decision in cases of high occupancy. The data center consists again of 500 physical servers, but in this case, 2000 VM requests have to be placed in each episode. The data center does not have enough capacity to serve all of them. Figure 3 shows the percentage of placed requests, calculated as the mean of the last 5 iterations. The heuristic-based agent (Round Robin) only manages to

accommodate 25% of the requests. This is inherent to the nature of Round Robin algorithm, that tries to spread out the load across different nodes. This naturally leads to resource fragmentation and limits the amount of resources that can be placed in a data center. In contrast, RL-based agents quickly learn a policy that tries to maximize the resource utilization. Both state-of-the-art baseline methods, PPO and PG, achieve a higher rate of successful placements in contrast to the heuristic agent, 89% and 91% respectively. Thanks to its parallel architecture, *ImpalaE* agent is able to explore more scenarios in a shorter amount of time and thus, further train the policy to score the highest placement rate, 94% of the 2000 VM requests within the same edge data center.

5.2. Agent scalability

The single learner-multiple worker architecture makes the proposed agent highly scalable, which in turns allows for faster convergence. The next experiment explores the impact of the number of workers in the training process. The scenario uses 500 physical servers and 1000 VM requests per episode, and compares the performance of PPO and *ImpalaE* (see figure 4). As expected, PPO shows the slowest convergence rate, easily surpassed by *ImpalaE* with a single worker. At its core, *ImpalaE* relies on several workers interacting with the environment and gathering as much information as possible, that is, they explore different data center scenarios and placement decisions and record the outcome of such decision (did it improved the request acceptance?). For this reason, increasing the number of works naturally improves the placement policy (higher reward) and leads to an earlier convergence. In this particular case, *ImpalaE* achieves the best results with 4 works. However, it is interesting to note that an additional worker (up to 5) actually achieves a slightly worse policy, which might be due to high variance in the sampling. We leave for future work the deeper analysis of the algorithm stability during training.

A well-known drawback of RL-based agents is their extremely long times (hours) needed to converge to an optimal policy, which makes it unfeasible to deploy such agent in a production environment. This experiment analyses the overall training time of the agent for a data center composed of 500 physical servers. As figure 5 shows, the baseline method, PPO, requires around 37 minutes of total training time. In contrast, the parallel architecture of *ImpalaE* allows it to further reduce the training time to only 4.4 minutes with 4 workers. This is especially appealing feature for highly dynamic environments, where the workload request patterns and resource usage change abruptly.

6. Conclusions and future work

Edge computing was born as an extension of widely used Cloud computing, with the differences that computing resources are located closer to the end-user and this is imperative for latency-critical applications. Edge computing providers face an additional challenge when making an optimal resource management of their data centers with reduced capacity, while trying to meet the client demand. This paper introduces *ImpalaE*, an agent based on Deep Reinforcement Learning, specially designed to optimize resource usage at the edge. It leverages Proximal Policy Optimization for finding the best placement policy for applications in edge data centers. It is also based on the IMPALA architecture, an asynchronous paradigm composed of one learner

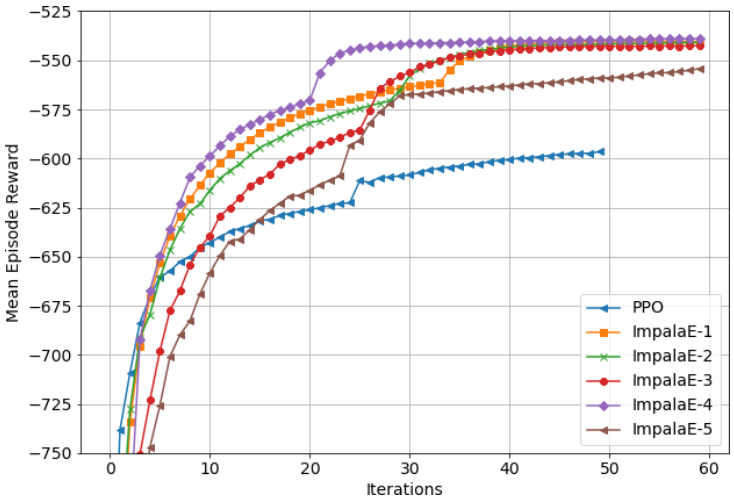

Figure 4: Mean reward per episode during training time.

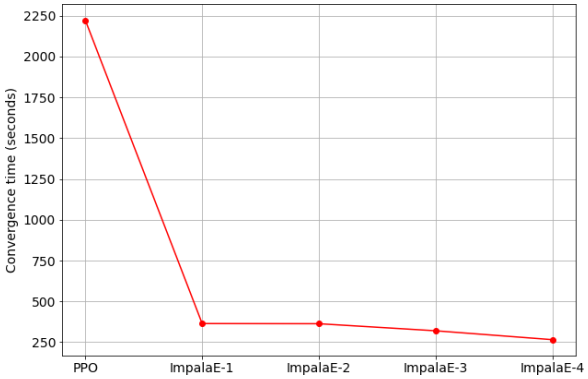

Figure 5: Convergence time.

and multiple parallel workers that speed up the convergence, even with reduced amount of data. The paper also introduces modeling of the edge computing environment as a Markov Decision Process with a bi-objective reward function specially designed to squeeze maximum performance. The validity of *ImpalaE* is assessed in a simulated environment considering VM requests based on real Microsoft Azure traces and considering CPU, memory, disk and network requirements.

The full potential of IMPALA architecture is yet to be explored. It has demonstrated higher performance with less data and ability to transfer information among tasks [3]. One natural

extension would be to expand *ImpalaE* to multiple data centers, that learn an optimal policy per data center, but also benefit from asynchronously exchanging information among different agents. However, there is also a need for deeper experimentation about the training stability for larger number of workers.

The current environment model takes into account the network bandwidth needs of each application. However, it could be further extended to consider the communication pattern among different nodes or VMs within the application. The reward function could be augmented with other objectives, such us application latency experienced by end-user or the data center energy utilization.

References

[1] Choo, K.K.R., Lu, R., Chen, L. and Yi, X., 2018. A foggy research future: Advances and future opportunities in fog computing research.

[2] Du, B., Wu, C. and Huang, Z., 2019. Learning resource allocation and pricing for cloud profit maximization. *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 33, pp.7570–7577.

[3] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I. and Others, 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *International Conference on Machine Learning*. PMLR, pp.1407–1416.

[4] Fan, Q. and Ansari, N., 2017. Cost aware cloudlet placement for big data processing at the edge. *2017 IEEE International Conference on Communications (ICC)*. IEEE, pp.1–6.

[5] Hadary, O., Marshall, L., Menache, I., Pan, A., Greeff, E.E., Dion, D., Dorminey, S., Joshi, S., Chen, Y., Russinovich, M. and Others, 2020. Protean: VM Allocation Service at Scale. *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. pp.845–861.

[6] Khan, W.Z., Ahmed, E., Hakak, S., Yaqoob, I. and Ahmed, A., 2019. Edge computing: A survey. *Future Generation Computer Systems*, 97, pp.219–235.

[7] Li, Y. and Wang, S., 2018. An energy-aware edge server placement algorithm in mobile edge computing. *2018 IEEE International Conference on Edge Computing (EDGE)*. IEEE, pp.66–73.

[8] Liu, J., Mao, Y., Zhang, J. and Letaief, K.B., 2016. Delay-optimal computation task scheduling for mobile-edge computing systems. *2016 IEEE International Symposium on Information Theory (ISIT)*. IEEE, pp.1451–1455.

[9] Lorido-Botran, T. and Bhatti, M.K., 2021. ImpalaE: Towards an optimal policy for efficient resource management at the edge. *CEUR Workshop Proceedings*, 2850, pp.71–82. Available from: http://ceur-ws.org/Vol-2850/paper5.pdf.

[10] Market Insights Reports, 2021-02. Global Edge Computing Market Size, Status And Forecast 2020-2026.

[11] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*. PMLR, pp.1928–1937.

[12] Mondal, S., Das, G. and Wong, E., 2018. CCOMPASSION: A hybrid cloudlet placement

framework over passive optical access networks. *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, pp.216–224.

[13] Puthal, D., Obaidat, M.S., Nanda, P., Prasad, M., Mohanty, S.P. and Zomaya, A.Y., 2018. Secure and sustainable load balancing of edge data centers in fog computing. *IEEE Communications Magazine*, 56(5), pp.60–65.

[14] Qiu, X., Liu, L., Chen, W., Hong, Z. and Zheng, Z., 2019. Online deep reinforcement learning for computation offioading in blockchain-empowered mobile edge computing. *IEEE Transactions on Vehicular Technology*, 68(8), pp.8050–8062.

[15] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[16] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Others, 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), pp.484–489.

[17] Tao, Z., Xia, Q., Hao, Z., Li, C., Ma, L., Yi, S. and Li, Q., 2019. A survey of virtual machine management in edge computing. *Proceedings of the IEEE*, 107(8), pp.1482–1499.

[18] Trace, 2021-02. Azure Public Dataset. Available from: https://github.com/Azure/AzurePublicDataset.

[19] Wang, Z., Gwon, C., Oates, T. and Iezzi, A., 2017. Automated cloud provisioning on aws using deep reinforcement learning. *arXiv preprint arXiv:1709.04305*.

[20] Zhang, S., Wu, T., Pan, M., Zhang, C. and Yu, Y., 2020. A-SARSA: A Predictive Container Auto-Scaling Algorithm Based on Reinforcement Learning. *2020 IEEE International Conference on Web Services (ICWS)*. IEEE, pp.489–497.

[21] Zhu, T., Shi, T., Li, J., Cai, Z. and Zhou, X., 2018. Task scheduling in deadline-aware mobile edge computing systems. *Ieee internet of things journal*, 6(3), pp.4854–4866.