RISCLESS: A Reinforcement Learning Strategy to Guarantee SLA on Cloud Ephemeral and Stable Resources

Sidahmed Yalles, Mohamed Handaoui, Jean-Emile Dartois, Olivier Barais, Laurent d'Orazio, Jalil Boukhobza

RISCLESS: A Reinforcement Learning Strategy to Guarantee SLA on Cloud Ephemeral and Stable Resources

SidAhmed Yalles*‡, Mohamed Handaoui*‡, Jean-Emile Dartois*†,
Olivier Barais*†, Laurent d'Orazio*†, Jalil Boukhobza*‡

*b<>com Institute of Research and Technology, †Univ. Rennes, Inria, CNRS, IRISA,
‡ENSTA Bretagne, Lab-STICC, CNRS, UMR 6285, F-29200 Brest, France

***Abstract*—In this paper, we propose RISCLESS, a Reinforcement Learning strategy to exploit unused Cloud resources. Our approach consists in using a small proportion of stable on-demand resources alongside the ephemeral ones in order to guarantee customers SLA and reduce the overall costs. The approach decides when and how much stable resources to allocate in order to fulfill customers' demands. RISCLESS improved the Cloud Providers (CPs)' profits by an average of 15.9% compared to past strategies. It also reduced the SLA violation time by 36.7% while increasing the amount of used ephemeral resources by 19.5%.**

***Index Terms*—Cloud, Resource Allocation, Ephemeral Resources, Stable Resources, SLA, Reinforcement learning**

I. Introduction

In order to reduce the cost of operating underutilized resources in a data-center, Cloud Providers (CPs) can reclaim the unused resources from *regular customers* (the ones who reserved these resources) to (re)sell it at a lower price to other customers (let us call them *ephemeral customers*). These reclaimed resources are by nature volatile. The resale of such resources must meet the ephemeral customers' expectations in terms of SLA. If the SLA is violated, CPs may be subject to penalties. Deploying applications on volatile resources while guaranteeing SLA is still a challenge [1]–[5]. Indeed, volatile resources can be lost and returned to their owner (the *regular customers*) in the event that their applications see their resource requirements increase. This change in regular customers' application behavior is hard to predict [6]–[8].

Different strategies were proposed to improve resource utilization and guarantee customers SLA on ephemeral resources. Some strategies [1], [3]–[5], [9]–[11] solely rely on ephemeral resources. They leave a proportion of those resources unused, called a safety margin, to absorb the sudden increase in regular customers' application demand, which decreases the amount of reclaimable resources. Other strategies [12]–[16] combine stable resources with volatile ones to guarantee customers' SLA. Nonetheless, they mainly focus on Amazon Spot Instance[1] which is less volatile than the reclaimed resources. Thus guaranteeing SLA while increasing the CPs' profits is a real challenge.

We argue that Machine Learning (ML) can be used to determine when and how much stable resources to allocate on top of the ephemeral ones (stable resources volume needs to be minimized). Specifically, we used Reinforcement Learning (RL) due to the limitations of classical solutions. In fact, most of the solutions [3], [12], [13] are centered on the parametric improvement and optimization of allocation strategies based on heuristics. Those are sometimes difficult and time-consuming to (re)configure. Above all, the solutions are not flexible to changes. To solve this problem, researchers have proposed methods based on ML with models that are capable of autonomously learn policies. More specifically, RL approaches have been proposed for task scheduling [17], [18] and resource allocation [19]–[21]. Although these studies do not consider ephemeral resources, they show that RL is indeed a promising choice to solve similar problems.

This paper proposes an approach to Cloud resource allocation that improves the utilization of ephemeral resources while guaranteeing SLA. Our solution computes the volatility rate of resources using past utilization traces. It then captures information, namely *i)* customers allocation request, *ii)* the amount of stable and ephemeral resources available and *iii)* the volatility rate of resources. It is used for the decision process of when to allocate ephemeral and stable resources in order to respond to customer's requests. All while increasing CPs' profits because stable resources are more expensive than ephemeral ones. Stable resources are useful mainly to temporarily absorb the high volatility of ephemeral resources. The solution also aims to reduce the possible penalties for violating the SLA.

Experimental evaluation on traces from three datacenters show that RISCLESS allows reducing SLA violation time by 36.7% on average compared to other strategies. The use of stable resources allows RISCLESS to compensate for the possible loss of allocated ephemeral resources.

II. RISCLESS : A ReInforcement Learning Strategy to Guarantee SLA

A. Architecture overview

Fig.1 presents an overview of the architecture that deploys our solution called RISCLESS (**ReI**nforcement Leaning **S**trategy to Guarantee SLA on **CL**oud **E**phemeral and **S**table Re**S**ources). There are three main actors:

Farmers: datacenter owners, that seek to reduce their TCO by offering unused resources to customers. We suppose that these farmers have stable resources that could be allocated on-demand with higher costs compared to the unused resources.

[1]https://aws.amazon.com/ec2/spot/

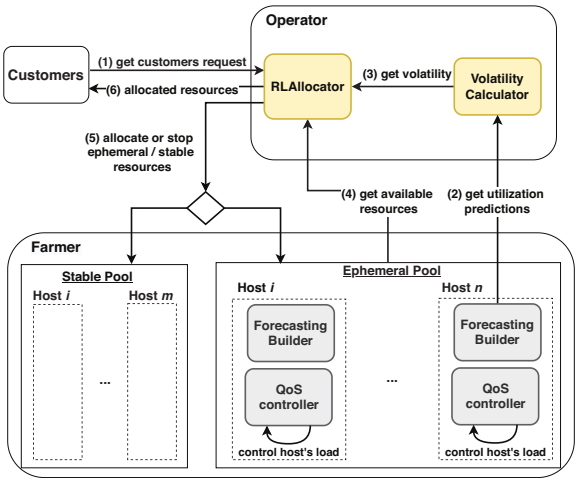


Fig. 1: Overview RISCLESS architecture and modules

Customers: we focus on customers that request ephemeral cloud resources at a lower cost (*i.e.,* ephemeral customers).

Operator: the interface between farmers and customers. They aim at minimizing farmers' TCO by offering unused resources to customers with SLA requirements.

Globally, the architecture is composed of ephemeral and stable resources. Each of these two resource pools is formed by a set of hosts. The hosts forming the ephemeral pool deploy the following two modules:

Forecasting Builder [2]: it predicts the next 24 hours of future resource utilization at a host-level. The amount of available resources is then inferred from hosts capacities.

QoS Controller [3], [5]: it ensures that the SLA of regular customers who have reserved resources on the host are respected. It does so by reducing the utilization of ephemeral resources if their owners (*i.e.,* regular customers) need them. The Operator deploys two modules (our contribution):

Volatility Calculator: this module is used to reduce the complexity of the resource allocation process. It summarizes the multiple points (*i.e.,* 24 hours) predictions from the Forecasting Builder into a single value. This value represents the volatility rate (*i.e.,* probability) of losing ephemeral resources.

RLAllocator: represents the decision-maker of our solution. It is based on an RL algorithm to decide when to allocate ephemeral and stable resources. The module aims at allocating the resources requested by customers with SLA requirements while maximizing CPs' profits.

The architecture in Fig.1 illustrates the workflow and communication between the actors and modules. The workflow starts with (1) the customers requesting resources for allocation through the *Customers* interface. The request is received by the *Operator* who transfers it to the *RLAllocator* module. The *Volatility Calculator* module then (2) retrieves the predictions of ephemeral resources and calculates their volatility rate. The module sends (3) the volatility rate to the *RLAllocator* module. The *RLAllocator* module also (4) receives the available ephemeral resource capacities. Once all the data have been collected, the *RLAllocator* module (5) decides on the pool of resources to allocate namely ephemeral or stable resources. Finally, The Operator (6) informs the customers of the allocated resources.

TABLE I: Forecasting Builder Example

t	$\widehat{y}_{cpu}$	y_{cpu}	$e_{cpu} = \widehat{y}_{cpu} - y_{cpu}$	$\widehat{y}_m$	y_m	$e_m = \widehat{y}_m - y_m$	z
0	30%	60%	-30%	40%	60%	-20%	1
1	40%	30%	10%	53%	50%	3%	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
479	41%	52%	-11%	45%	40%	5%	1

B. Volatility Calculator

The Forecasting Builder predicts future resource utilization for every host. Having the hosts' predictions for the next 24 hours as input for the RLAllocator is costly as the size increases with the number of hosts. Through the Volatility Calculator module, our goal is to reduce the verbose information and only provide a single value that represents the volatility rate of losing resources in the ephemeral pool.

The volatility rate provides a summary of prediction errors to the RLAllocator. The latter can then make allocation decisions based on the provided information. The module receives as input both the past prediction and utilization of resources during a Δt window (*e.g.,* 24 hours). The module then computes the volatility rate $p \in [0, 1]$ of the ephemeral pool. Finally, the module outputs the value for the RLAllocator.

Table I shows an example of traces with predictions from the Forecasting Builder for a time window of Δt=24h with a 3-minute sampling period (which makes 480 periods). It contains the following for both the CPU and memory metrics:

- Actual measures of utilization: y_{metric}
- Predictions of future resource utilization: $\widehat{y}_{metric}$
- Prediction errors: $e_{metric} = \widehat{y}_{metric} - y_{metric}$

The volatility rate represents the probability of underestimating the amount of resource utilization. In other words, resources are lost if $\widehat{y}_{metric} < y_{metric}$ and the amount of resources lost is proportional to the prediction error.

To calculate this probability, a random variable z is used where z_t represents whether the predictions underestimated the CPU or memory usage at time t. Table I shows the values that the variable z takes according to its definition. It is set to '1' if the prediction is underestimated (*i.e.,* $e_{metric} < 0$). For example, at $t = 0$, the predicted CPU is $\widehat{y} = 30\%$ but the measured utilization is $y = 60\%$, thus the prediction error is $e_{cpu} = -30\% < 0$ and $z = 1$.

Assuming that the different measures are independent of each other, the variable z follows a Bernoulli's distribution of parameter p. With p being the probability of underestimation.

To estimate p, an empirical estimator $\widehat{p}$ is used. It is the mean over a Δt time window of the z values.

C. RLAllocator

It is the decision-maker, its objective consists in deciding when to allocate ephemeral and stable resources in order to maximize CPs' profits and reduce SLA violations.

This module was built using RL. It is formalized with the MDP composed of the quadruplet $< S, A, T, R >$ with: S

being the set of states of the environment, A the set of actions, T the transition function, and R the reward function.

*1) **Environment**:* Each of the two pools of resources is characterized by its available capacity in terms of CPU and memory. The ephemeral pool is further characterized by the volatility rate. Both types of resources have a cost of allocation. Ephemeral resources are less expensive than the stable ones (up to 90% in Amazon Spot Instance).

*2) **State space** $\mathbf{S}$:* At each time step t, the state is characterized by the customers' request, the quantity of ephemeral and stable resources allocated, the available capacity of the two types of resources, and finally the volatility rate of the ephemeral resources. It is defined as follows:

$S = \{res_{rem},\ res_{alloc}(e),\ res_{alloc}(s),\ res_{avail}(e),\ res_{avail}(s),\ p\}$

With res_{rem}, $res_{alloc}(e)$, $res_{alloc}(s)$, $res_{avail}(e)$, $res_{avail}(s) \in \mathbb{N}$ being respectively the amount of remaining resources to be allocated, ephemeral resources allocated, stable resources allocated, available ephemeral resources, available stable resources and $p \in [0, 1]$: the volatility rate.

*3) **Action space** $\mathbf{A}$:* At each time step t, the model can perform an action a, which is related to the allocation of resources for the customers. The set of actions are:

$A = \{a_1, a_2, a_3, a_4, a_5\}$

with: a_1: Allocate an ephemeral resource unit, a_2: Remove an ephemeral resource unit. a_3: Allocate a stable resource unit. a_4: Remove a stable resource unit. a_5: Do nothing.

A resource unit is defined as an amount of vCPU and memory that are allocated at the same time to a customer (*e.g.,* a resource unit of 2 vCPU and 8 GB).

*4) **Reward function** $\mathbf{R}$:* Our goal is to maximize CPs' profits from selling ephemeral resources while minimizing the use (thus the cost) of stable resources for ephemeral customers. We also seek to reduce SLA violations for regular customers. The reward function is defined for each state as: $r = res_{alloc}(e) \times CPE - res_{alloc}(s)\ \times$ $CPS - res_{rem}\ \times CPV$ with CPE being the cost per ephemeral resource unit CPS, the cost per stable resource unit CPV, the cost of SLA violation penalty.

Each ephemeral or stable resource has a cost per unit CPE and CPS. The SLA violation has a cost per violation CPV. In a state S, the reward function considers the total cost of the amount of ephemeral resources allocated $res_{alloc}(e)$ which is considered as the profit. The amount of stable resources $res_{alloc}(s)$ has to be minimized since its cost is higher than the ephemeral ones. SLA violation can occur in two cases: i) when losing an ephemeral resource, ii) when the customer requests are not met . In both cases, the remaining resource to allocate res_{rem} increases, hence increasing SLA penalties.

*5) **Model algorithm**:* To solve the Cloud resource allocation problem, we train the RL agent using the Deep Q-Network (DQN) algorithm [22]. DQN is used to approximate the Q-values using neural networks with a single function (called Q network). Since the state representation of the allocation problem is too large, DQN can approximate values for the Cloud states that have never been encountered during the learning process. Algorithm 1 represents the pseudo-code of allocating Cloud ephemeral and stable resources using DQN.

The algorithm starts by initializing (line 1) the configuration of the agent's model. Then, it initializes a buffer that stores previous resource allocation experiences (line 2). The buffer is used to improve the learning process of the agent. Each experience contains the state (*e.g.,* allocated amount of ephemeral and stable resources), action (*e.g.,* allocate stable resource), reward (*e.g.,* SLA violation penalty), and the next state of the Cloud environment. The agent starts by receiving the amount of resources to allocate (line 3). Then it receives the available amount of resources for the ephemeral and stable pools (lines 5-7). The agent then makes either a random resource allocation decision (*i.e.,* action) or the best one according to a probability ϵ (line 8). The random selection of actions is necessary since initially, the agent does not have any experience. Afterward, the agent fetches both the reward and the new environment state (line 9). This current resource allocation experience is stored in the buffer (line 10) used for the learning process (lines 11-13). The decision process is repeated as long as there are still resources available to be allocated for customers.

Algorithm 1: Pseudo-code of the used DQN

```
1.  agent = initialize_DQN_model();
2.  experiences = initialize_experience_buffer();
3.  res_rem = get_remaining_resources_to_allocate();
4.  while res_rem > 0 do // remaining resources to
    allocate
5.      res_alloc = get_allocated_resources();
6.      res_avail = get_available_resources();
7.      p = compute_volatility_rate();
8.      select a= { random action   with probability ε
                  { best action     else
9.      reward = observe_reward_value();
10.     experiences.add_current_experience()
11.     if should_update then
12.         agent.update(experiences);
13.     end
14.     res_rem = get_remaining_resources_to_allocate();
15. end
```

III. Experimental evaluation

We try to answer the following Research Questions (RQ): **RQ1:** What is the overall performance of RISCLESS in terms of resource utilization, SLA violations, and CPs' profits? **RQ2:** How many on-demand stable resources does RISCLESS use on top of the ephemeral ones to reduce SLA violations ?

A. Experimental setup

*1) **Datasets**:* the traces used were extracted from three datacenters. One is from a University, and two from Private Companies labeled PC-1 and PC-2. The traces were recorded over 6 months with a 3-minute sampling period [2], [3], [11].

*2) **Resource allocation approaches**:* RISCLESS was compared to the following approaches:

Fixed: [3]: This approach uses a static safety margin percentage of 5% selected empirically from different datasets.

Scavenger [4]. The mean and standard deviation of resource utilization history is used to compute a dynamic safety margin.

ReLeaSER [11]: It uses Reinforcement Learning to select a dynamic safety margin according to resource prediction errors.

TABLE II: Summary information of each datacenter

Datacenter	Number of hosts	CPU (cores)	RAM (TB)
PC-1	9	120	1.2
PC- 2	27	230	3.8
University	6	72	1.5

*3) **Implementation**:* Experiments were performed on an ad-hoc simulator. RISCLESS was implemented using Keras v. 2.3.1 and TensorFlow GPU v. 1.14.0. The Mean Square Error was used as the error function. The training was performed on 80% of PC-2 traces, while 20% were used for testing.

B. Evaluation metrics

We used the following metrics: **1) Total profits:** which is the cost related to the profits minus the cost of SLA violations and the cost related to the on-demand stable resources.**2) SLA violation time:** it is the cumulative time during which the SLA was violated. **3) Amount of reclaimed ephemeral resources:** it is the cumulative amount of ephemeral resources that were used without affecting the SLA of customers.

The evaluation of the first two metrics is based on a real economical model from Amazon AWS. It comprises:

Resource costs: based on Amazon AWS instance type *t2.large* that is 2 vCPU and 8 GB of memory (ephemeral instance: 0.0317 $/hour, stable instance: 0.0928 $/hour).

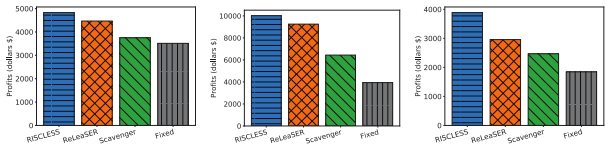
SLA violation: the penalty is calculated as a discount on the profit related to the sold instances. The discount percentage is based on the cumulative violation time over one day:

- Between 15 and 120 minutes: 10% discount
- Between 120 and 720 minutes: 15% discount
- More than 720 minutes: 30% discount

*1) **RQ-1. Overall performance of RISCLESS**:* Fig.2 shows the total profits of RISCLESS over the 6 months traces for each datacenter. We observe that for the three datacenters, the Fixed strategy generates the least profits. We also observe that ReLeaSER performs better than Scavenger by an average of 27.6%. Finally, RISCLESS generates the highest profits compared to other approaches. It improves the profits compared to ReLeaSER by 8%, 8.3%, and 31.5% for PC-1, PC-2, and University respectively. These results are explained by the SLA violation time and the amount of reclaimed resources.

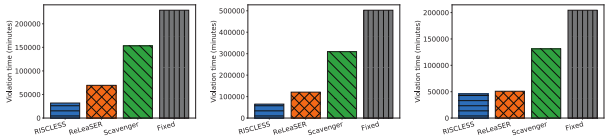
Fig.3 shows the cumulative time during which the SLA is violated. We observe that RISCLESS violates SLA the less. It reduces the cumulative violation time when compared to ReLeaSER by 54%, 46.2% and 10% for PC-1, PC-2, and University respectively. These results show that the utilization of stable resources can decrease the SLA violation time. This partly explains the improvements in the profits seen previously.

Fig.4 shows the average amount of used ephemeral resources per day for each datacenter. This amount is measured as the cumulative number of allocated resource units for each time step throughout the day. The red line shows the maximum reclaimable resources. We observe that the Fixed approach utilizes the least ephemeral resources which can be explained by the safety margin used. ReLeaSER uses around 2% fewer resources when compared to Scavenger but still manages to generate more profits. This is mainly due to the reduction in SLA violation time. We also observe that RISCLESS uses more ephemeral resources. When compared to Scavenger, it improves the utilization by 12.8%, 10.9%, and 34.8% for PC-1, PC-2, and University respectively. The approaches that use a safety margin reduce the amount of usable resources to avoid SLA violations. However, using stable resources to absorb the potential loss of volatile resources allows RISCLESS to optimize its utilization. RISCLESS uses 92%, 98%, and 93% of the maximum reclaimable resources for PC-1, PC-2, and University while having the least of SLA violations.



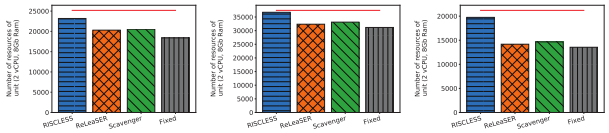
(a) PC 1 (b) PC 2 (c) Univ.

Fig. 2: Total CPs' profits



(a) PC 1 (b) PC 2 (c) Univ.

Fig. 3: Cumulative violation time of SLA



(a) PC 1 (b) PC 2 (c) Univ.

Fig. 4: Average per day amount of ephemeral resources

*2) **RQ-2. Percentage of stable resources**:* we analyze results concerning the percentage of stable resources used. The goal is to extract some of the environment variables (*e.g.,* volatility rate) that affect the utilization of stable resources. Table III specifies, for each datacenter, the average volatility rate per day of the ephemeral resources, as well as the percentage of stable resources used compared to the total allocated resources for ephemeral customers.

We can observe that PC-1 has the highest volatility rate and the highest percentage of stable resources. Meanwhile, PC-2 has the lowest volatility rate and stable resources. RISCLESS uses less than 10% of stable resources for all the tested datacenters. It can be observed that the more volatile the resources of a datacenter, the more stable resources are used.

IV. Related Work

Safety margin-based approaches: safety margin was used in [1], [3], [5], [9], [10] with a fixed percentage of safety

TABLE III: Average volatility rate and % of stable resources

	PC-1	PC-2	University
Volatility rate	0.83	0.69	0.75
Stable resources used (%)	9.21%	4.63%	8.33%

margin. Even though the fixed method does reduce SLA violations, it can be improved considerably alongside resource utilization since customers' workloads are not stable. Hence, a dynamic safety margin was used in Scavenger [4] and ReLeaSER [11]. It improved the utilization of ephemeral resources while reducing customers' SLA violations. However, when resource volatility is high, these strategies may not perform well. Indeed, the higher the volatility, the larger the safety margin, the less ephemeral resources are exploited.

Stable and ephemeral resources: other studies [12]–[16] tried to improve customers SLA by utilizing stable on-demand resources on top of the ephemeral ones. The stable resources can be used for saving data in the case of data processing applications. It can also be useful for running prioritized jobs that have to be otherwise re-executed due to the lost resources. However, the aforementioned solutions mainly focus on Amazon Spot Instance which is less volatile than the reclaimed resources. Furthermore, the customers using these resources receive a notification prior to the interruption. This signal can be used as a convenient moment for allocating stable resources. In addition, the use cases of these solutions are generally limited to data processing applications.

V. Conclusion

We proposed RISCLESS, a strategy that makes it possible to exploit ephemeral resources while reducing SLA violations. Our approach is based on RL as a decision-making model. It combines ephemeral resources with on-demand stable resources in order to offer SLA guarantees while reducing costs. The experimental evaluation results showed that RISCLESS had allowed for more thorough exploitation of ephemeral resources with a reduction in SLA violations, which significantly increased CPs' profits from the resale of resources.

Acknowledgment

This work was supported by the Institute of Research and Technology b-com, funded by the French government through the ANR Investment referenced ANR-A0-AIRT-07.

References

[1] Y. Zhang, G. Prekas, G. M. Fumarola, M. Fontoura, Í. Goiri, and R. Bianchini, "History-based harvesting of spare cycles and storage in large-scale datacenters," in *12th USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 755–770.

[2] J.-E. Dartois, A. Knefati, J. Boukhobza, and O. Barais, "Using quantile regression for reclaiming unused cloud resources while achieving sla," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2018, pp. 89–98.

[3] J.-E. Dartois, H. B. Ribeiro, J. Boukhobza, and O. Barais, "Cuckoo: Opportunistic mapreduce on ephemeral and heterogeneous cloud resources," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 396–403.

[4] S. A. Javadi, A. Suresh, M. Wajahat, and A. Gandhi, "Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments," in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 272–285.

[5] M. Handaoui, J.-E. Dartois, L. Lemarchand, and J. Boukhobza, "Salamander: a holistic scheduling of mapreduce jobs on ephemeral cloud resources," in *The 20th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2020, pp. 320–329.

[6] J. Cao, J. Fu, M. Li, and J. Chen, "Cpu load prediction for cloud environment based on a dynamic ensemble model," in *Software: Practice and Experience*, 2014, pp. 793–804.

[7] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," in *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 2009, p. 2009.

[8] J.-E. Dartois, J. Boukhobza, A. Knefati, and O. Barais, "Investigating machine learning algorithms for modeling ssd i/o performance for container-based virtualization," in *IEEE transactions on cloud computing*, 2019, pp. 1–15.

[9] L. Zhao, Y. Yang, K. Zhang, X. Zhou, T. Qiu, K. Li, and Y. Bao, "Rhythm: component-distinguishable workload deployment in datacenters," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–17.

[10] T. Patel and D. Tiwari, "Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 193–206.

[11] M. Handaoui, J.-E. Dartois, J. Boukhobza, O. Barais, and L. d'Orazio, "Releaser: A reinforcement learning strategy for optimizing utilization of ephemeral cloud resources," in *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2020.

[12] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang, "Moon: Mapreduce on opportunistic environments," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 95–106.

[13] Y. Yan, Y. Gao, Y. Chen, Z. Guo, B. Chen, and T. Moscibroda, "Tr-spark: Transient computing for big data analytics," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016, pp. 484–496.

[14] Y. Yang, G.-W. Kim, W. W. Song, Y. Lee, A. Chung, Z. Qian, B. Cho, and B.-G. Chun, "Pado: A data processing engine for harnessing transient resources in datacenters," in *Proceedings of the Twelfth European Conference on Computer Systems*, 2017, pp. 575–588.

[15] P. Sharma, D. Irwin, and P. Shenoy, "Portfolio-driven resource management for transient cloud servers," in *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2017, pp. 1–23.

[16] L. Lin, L. Pan, and S. Liu, "Backup or not: An online cost optimal algorithm for data analysis jobs using spot instances," in *IEEE Access*, 2020, pp. 144 945–144 956.

[17] F. Li and B. Hu, "Deepjs: Job scheduling based on deep reinforcement learning in cloud data center," in *Proceedings of the 2019 4th International Conference on Big Data and Computing*, 2019, pp. 48–53.

[18] S. Liang, Z. Yang, F. Jin, and Y. Chen, "Job scheduling on data centers with deep reinforcement learning," in *arXiv preprint arXiv:1909.07820*, 2019.

[19] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow," in *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, 2011, pp. 67–74.

[20] A. Galstyan, K. Czajkowski, and K. Lerman, "Resource allocation in the grid using reinforcement learning," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, 2004, pp. 1314–1315.

[21] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 372–382.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," in *nature*, 2015, pp. 529–533.