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

Sidahmed Yalles, Mohamed Hendaoui, Jean-Emile Dartois, Olivier Barais, Laurent d’Orazio, Jalil Boukhobza

To cite this version:

Sidahmed Yalles, Mohamed Hendaoui, Jean-Emile Dartois, Olivier Barais, Laurent d’Orazio, et al.. RISCLESS: A Reinforcement Learning Strategy to Guarantee SLA on Cloud Ephemeral and Stable Resources. 2022 30th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Mar 2022, Valladolid, Spain. pp.83-87, 10.1109/PDP55904.2022.00021 . hal-03921309v2

HAL Id: hal-03921309
https://hal.science/hal-03921309v2
Submitted on 5 Jan 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
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 Instances 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 (Reinforcement Learning Strategy to Guarantee SLA on CLoud Ephemeral and Stable ReSources). 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.
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.

Volatility rate. The module sends (3) the volatility rate to the predictions of ephemeral resources and calculates their volatility rate. The module then (2) retrieves the available ephemeral resources that are requested by customers with SLA requirements. 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 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: $\hat{y}_{metric}$
| Prediction errors: $e_{metric} = \hat{y}_{metric} - y_{metric}$

The volatility rate represents the probability of underestimating the amount of resource utilization. In other words, resources are lost if $\hat{y}_{metric} < y_{metric}$ 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 $\hat{g} = 30\%$ but the measured utilization is $g = 60\%$, 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 $\hat{p}$ is used. It is the mean over a $\Delta t$ time window of the $z$ values.

### 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 $\Delta t$ 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: Forecasting Builder Example

<table>
<thead>
<tr>
<th>$t$</th>
<th>$y_{cpu}$</th>
<th>$y_{cpu}$</th>
<th>$e_{cpu} = \hat{y}<em>{cpu} - y</em>{cpu}$</th>
<th>$\hat{y}_{mem}$</th>
<th>$y_{mem}$</th>
<th>$e_{mem} = \hat{y}<em>{mem} - y</em>{mem}$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>30%</td>
<td>60%</td>
<td>-30%</td>
<td>40%</td>
<td>60%</td>
<td>-20%</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>40%</td>
<td>30%</td>
<td>10%</td>
<td>53%</td>
<td>50%</td>
<td>3%</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>479</td>
<td>41%</td>
<td>52%</td>
<td>-11%</td>
<td>45%</td>
<td>40%</td>
<td>5%</td>
<td>1</td>
</tr>
</tbody>
</table>
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 $S$: 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:

$$S = \{ \text{res}_{\text{rem}}, \text{res}_{\text{alloc}}(e), \text{res}_{\text{alloc}}(s), \text{res}_{\text{avai}}(e), \text{res}_{\text{avai}}(s), p \}$$

With $\text{res}_{\text{rem}}, \text{res}_{\text{alloc}}(e), \text{res}_{\text{alloc}}(s), \text{res}_{\text{avai}}(e), \text{res}_{\text{avai}}(s) \in \mathbb{N}$ 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 $A$: 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:

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

- $a_1$: 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 $R$: 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:

$$r = \text{res}_{\text{alloc}}(e) \times CP_{E} - \text{res}_{\text{alloc}}(s) \times CP_{S} - \text{res}_{\text{rem}} \times CP_{V}$$

with $CP_{E}$ being the cost per ephemeral resource unit $CP_{S}$, the cost per stable resource unit $CP_{V}$, the cost of SLA violation penalty.

Each ephemeral or stable resource has a cost per unit $CP_{E}$ and $CP_{S}$. The SLA violation has a cost per violation $CP_{V}$. In a state $S$, the reward function considers the total cost of the amount of ephemeral resources allocated $\text{res}_{\text{alloc}}(e)$ which is considered as the profit. The amount of stable resources $\text{res}_{\text{alloc}}(s)$ 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 $\text{res}_{\text{rem}}$ 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 $\epsilon$ (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. $\text{res}_{\text{rem}} = \text{get_remaining_resources_to_allocate}();$
4. while $\text{res}_{\text{rem}} > 0$ do // remaining resources to allocate
5. $\text{res}_{\text{alloc}} = \text{get_allocated_resources}();$
6. $\text{res}_{\text{avai}} = \text{get_available_resources}();$
7. $p = \text{compute_volatility_rate}();$
8. select $a = \begin{cases} \text{best action} & \text{else} \end{cases}$
9. reward = observe_reward_value();
10. experiences.add_current_experience()
11. if should_update then
12. agent.update(experiences); end
13. $\text{res}_{\text{rem}} = \text{get_remaining_resources_to_allocate}();$
14. 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.
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.

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 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 dataset. 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

### TABLE II: Summary information of each datacenter

<table>
<thead>
<tr>
<th>Datacenter</th>
<th>Number of hosts</th>
<th>CPU (cores)</th>
<th>RAM (TB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC-1</td>
<td>9</td>
<td>120</td>
<td>1.2</td>
</tr>
<tr>
<td>PC-2</td>
<td>27</td>
<td>230</td>
<td>3.8</td>
</tr>
<tr>
<td>University</td>
<td>6</td>
<td>72</td>
<td>1.5</td>
</tr>
</tbody>
</table>

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.

IV. RELATED WORK

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


### TABLE III: Average volatility rate and % of stable resources

<table>
<thead>
<tr>
<th></th>
<th>PC-1</th>
<th>PC-2</th>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volatility rate</td>
<td>0.83</td>
<td>0.69</td>
<td>0.75</td>
</tr>
<tr>
<td>Stable resources used (%)</td>
<td>9.21%</td>
<td>4.63%</td>
<td>8.33%</td>
</tr>
</tbody>
</table>