Learning fast is painful

Theo Bouganim, Andrea Araldo, Antoine Lavignotte, Nessim Oussedik, Gabriel Guez

Learning Fast is Painful: Reinforcement Learning for Edge Computing Allocation

Theo Bouganim, Andrea Araldo, Antoine Lavignotte, Nessim Oussedik, Gabriel Guez
Télécom SudParis - Institut Polytechnique de Paris
Palaiseau - France
firstname.lastname@telecom-sudparis.eu

***Abstract*—We study the problem of data-driven resource allocation in Multi-Tenant Edge Computing: a Network Operator (NO) owns resources at the Edge and dynamically allocates them to third party application Service Providers (SPs). The objective of the NO is to reduce its operational cost. Since SPs' traffic is encrypted, NO's allocation strategy is based solely on the amount of traffic measured.**

In this exploratory work, we solve this problem via Reinforcement Learning (RL). RL has mainly been intended to be trained in simulation, before applying it in real scenarios. We instead employ RL *online*, training it directly while optimizing resource allocation. An important factor, which we call *perturbation cost*, emerges in this case: in order to learn how to optimize a system, we need to perturb it and measure its reaction. While this perturbation cost has no physical meaning when training RL in simulation, it cannot be ignored when it is paid by the real system. We explore in this work the trade-off between perturbing a lot the system to learn faster to optimize the allocation, or learning slower to reduce the perturbation cost. In our case study, the resource we allocate is storage. We show results from simulation and make the entire code available as open-source.[1]

1. Introduction

In Multi-tenant Edge Computing (EC) [3] resources are scarce, thus the NO has to carefully allocate them among SPs in order to obtain potential benefits. Such allocation is challenging, since application resource usage may be complex and unknown to the NO, obfuscated behind end-to-end encryption and memory encryption. In other words, SPs are black boxes for the NO. In such scenario, Machine Learning data-driven (as opposed to model-based) approaches, solely based on monitored measurements, are the only viable option for resource allocation.

RL algorithms are usually intended to be trained in simulation, but writing a simulation can be time-consuming and difficult. Recently, some authors have trained RL directly on real systems [1]. However, in our scenario, the only way to learn the optimal allocation (or at least a good-enough allocation) is to *perturb* it and measure how the system changes. This perturbation inherently represents a cost (*perturbation cost*), which during simulation can be neglected, but that can be detrimental when training is performed in the real system.

In this work, we apply RL to allocate storage in order to minimize the inter-domain traffic for the NO. The perturbation cost is due to the fact that whenever new space is given to a SP, some traffic must be generated to place some content there. We show that the "learn as fast as possible" goal of generic machine learning approaches is not valid when applying data-driven optimization to a real system. On the contrary, a delicate trade-off between learning a good optimization strategy while avoiding large system perturbations must be pursued.

2. System model

A NO owns storage resources in an edge cluster [2] and aims to minimize the downstream traffic arriving from other Autonomous Systems (ASes). To do so, it allocates a total storage of K slots among P Service Providers (SPs). Each SP is a video streaming service, which caches its most popular objects (videos) in its allotted slots. As usual in the literature, one slot can store an object. The allocation is a vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_P)$, where each θ_p is the number of slots given to SP p and $\sum_{p=1}^{P} \theta_p \leq K$. The time is slotted and at any time intervals the NO may change allocation, giving Δ slots to one SP and $-\Delta$ slots to another. We assume that whenever SP p is given θ_p slots, it will cache there its most popular θ_p objects. We assume there are no other caches outside the edge, as advocated by previous work [4]. A user request for an object withing those popular ones is served directly by the edge, otherwise the object must arrive from another AS, constituting inter-domain traffic. At any time interval, we measure the *nominal cost* C_{nom} and the *perturbation cost* C_{pert}. The former is the miss stream, i.e., the amount of requests for objects that are not in the cache. C_{pert} occurs when an SP had θ_p and then it is given $\theta_p + \Delta$ slots. In this case, it has to download the Δ new objects. In this case, $C_{\text{pert}} = \Delta$. In our future work, we will consider smarter strategies, in which new objects are downloaded only after they are requested for the 1st time. We define the total cost as $C_{\text{tot}} = C_{\text{nom}} + C_{\text{pert}}$.

1. https:github.com/Ressource-Allocation/Cache-Allocation-Project

3. Reinforcement Learning Formulation

The main parameter of our formulation is $\Delta \in \mathbb{N}$, i.e., the number of slots by which we dynamically modify the storage allocation. Following Q-learning notation, the system is described by a set of *states*, which correspond to all the possible allocation vectors, i.e.,

$$\mathcal{S} = \left\{ \boldsymbol{\theta} = (\theta_1, \ldots, \theta_P) | \sum_{p=1}^{P} \theta_p \leq K, \theta_p \text{ multiple of } \Delta \right\}$$

Note that only when K is a multiple of Δ we can exploit all the slots. At any time interval, based on the measured C_{nom}, the NO takes an action to modify the allocation. When in state $\boldsymbol{\theta}$, the space of possible actions for the NO is:

$$\mathcal{A}_{\boldsymbol{\theta}} = \{ \mathbf{a} = \Delta \cdot \mathbf{e}_p - \Delta \cdot \mathbf{e}_{p'} | \boldsymbol{\theta} + \mathbf{a} \in \mathcal{S}, p, p' = 1, \ldots, P \}$$

where $\mathbf{e}_p$ is the p-th element of the standard basis of $\mathbf{R}^p$. To choose the action, we resort to a simple ϵ-greedy policy and SARSA algorithm. Following the notation in §5.4 and §6.4 of [5], we set $\epsilon = 0.2, \gamma = 0.4, \alpha = 0.9$. We set SARSA's "instantaneous reward" equal to $-C_{\text{nom}}$, measured at each time interval (using $-C_{\text{tot}}$ instead of $-C_{\text{nom}}$, convergence was too slow). Note that the more Δ, the more aggressive is learning: C_{nom} may converge faster, thanks to the reduced granularity of actions, but suffering higher C_{pert}.

4. Performance Evaluation

We consider 3 SPs. In line with the literature, users generate 10^3 requests/sec, each directed to SP p with probability q_p. The catalog of any SP p contains 10^6 videos, whose popularity follows a Zipf law with exponent s_p. The total cache storage at the Edge is $K = 10^3$ slots, each containing one video. The values for q_p, s_p are:

	SP $p=1$	SP $p=2$	SP $p=3$
q_p	0.7	0.25	0.05
s_p	0.8	1.0	1.2

The RL action modifying the allocation is taken every 1 sec. In Fig. 1, we compare the costs from the RL allocation to two static allocations: (i) the optimal cost-minimizing allocation hypothetically computed by an oracle who can observe and predict all the requests and an (ii) initial allocation, which we choose purposely far from the optimum, to verify that RL is able to dynamically correct it. Obviously, there is no perturbation cost for static allocations. Curves are smoothed with 10-minutes windows and normalized by the total amount of requests. Observe that with $\Delta = 100$, C_{nom} converge fast, but we impose large perturbations, resulting in a high C_{tot}. Note that we purposely avoid to decrease the Δ during time, as we want our policy to be responsive to exogenous changes (for instance of content popularity, of overall request load, etc.). It is preferable instead to choose a perturbation Δ which is small enough to avoid high C_{pert} and large enough to allow for fast convergence, and thus prompt adaptivity to exogenous changes. Unfortunately, there is no value of Δ that fits best all the cases: Fig. 2 shows that the impact of C_{pert} is exacerbated when the system load is lower than 10^3req/sec, requiring even smaller perturbations.

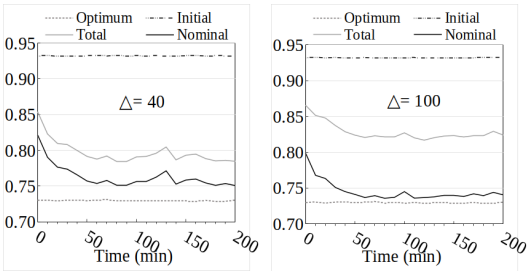

Figure 1. Cost over time.

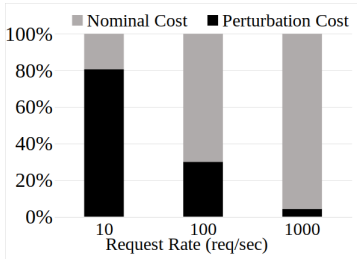

Figure 2. Cost Breakdown, with $\Delta = 40$.

5. Future work

Since this is a preliminary work, we make the assumptions and the limitations explicit. (i) The request load is stationary, i.e., the object popularity distribution does not change with time. We will need to extend the definition of state if we want to apply RL to a general case. (ii) Under convexity assumptions, the storage allocation problem can be solved by Stochastic Optimization (SO) algorithms [2], which we plan to compare against. However, we plan to apply RL to more complex scenario, where we will allocate memory, CPU, bandwidth and the objective function will also include quality of service or quality of experience indicators. In such complex cases, convexity cannot be assumed, which justifies resorting to RL. (iii) The results of this work come from simulation. We are currently working obtaining them on a real testbed.

References

[1] Mirhoseini A. et al. "Device placement optimization with reinforcement learning". In: *ICML*. 2017.

[2] A. Araldo et al. "Caching Encrypted Content Via Stochastic Cache Partitioning". In: *IEEE/ACM Transactions on Networking* 26.1 (2018), pp. 548–561.

[3] A. Araldo et al. "Resource Allocation for Edge Computing with Multiple Tenant Configurations". In: *ACM/SIGAPP SAC*. 2020.

[4] Fayazbakhsh et al. "Less Pain Most of the Gain: Incrementally Deployable ICN". In: *ACM Sigcomm*. 2013.

[5] Sutton R. et al. *Reinforcement Learning: An Introduction*. MIT Press, 2017.