No Regret in Cloud Resources Reservation with Violation Guarantees

Nikolaos Liakopoulos[1,2], Georgios Paschos[1], Thrasyvoulos Spyropoulos[2]

[1]Mathematical and Algorithmic Sciences Lab, FRC, Huawei Technologies SASU, email: firstname.lastname@huawei.com
[2]EURECOM, Sophia-Antipolis France, email: spyropou@eurecom.fr

***Abstract*—This paper addresses a fundamental challenge in cloud computing, that of learning an economical yet robust reservation, i.e. reserve just enough resources to avoid both violations and expensive over provisioning. Prediction tools are often inadequate due to observed high variability in CPU and memory workload. We propose a novel model-free approach that has its root in online learning. Specifically, we allow the workload profile to be engineered by an adversary who aims to harm our decisions, and we investigate a class of policies that aim to minimize regret (minimize losses with respect to a baseline static policy that knows the workload sample path). Then we propose a combination of the Lyapunov optimization theory [1] and a linear prediction of the future based on the recent past, used in learning and online optimization problems, see [2]. This enables us to come up with a no regret policy, i.e., a policy whose cost difference to the benchmark and violation constraint residual both grow sublinearly in time, and hence become amortized over the horizon. Our policy has then "no regret", and eventually learns the minimum cost reservation subject to a time-average constraint for violations.**

I. Introduction

A fundamental challenge in cloud computing is to reserve just enough resources (e.g. memory, CPU, and bandwidth) to meet application runtime requirements [3]. We desire reservations that accurately meet the requirements: resource over provisioning causes excessive operation costs, while under provisioning may severely degrade service quality, causing interruptions and real-time deployment of extra resources, which costs heavily [4]. The problem resembles the well-known *newsvendor model* [5], where we seek an inventory level that maximizes the vendor revenue versus a forecast demand. In cloud computing, however, the common assumption of demand predictability does not hold. Recent experimentation in a Google cluster [6] shows that the profile of cloud resources exhibits highly non-stationary behaviour, and prediction is very difficult, if not impossible. Furthermore, in the increasingly relevant scenario of edge computing, the workload is expected to vary quickly with geography, mobility, and user application trends, and therefore its fluctuations will be even more unpredictable. All these motivate the approach in this paper; *to design a model-free online reservation policy for cloud computing using ideas from machine learning*.

A concern with machine learning approaches is that their exploration phase combined with occasional unpredictability, may lead to an unforeseen violation of an important constraint. In our case, reserving fewer resources than needed for long time periods may potentially mount a serious threat on the operation of the cloud system. Therefore, apart from the demand unpredictability, we are faced with the extra challenge of guaranteeing that the average resource violations will not exceed a pre-defined threshold. To address both aforementioned challenges at once, we cast the problem of resource reservation in the setting of constrained-*Online Convex Optimization* (OCO), seeking to find a no regret policy with violation guarantee. This is an extension of the standard machine learning framework OCO, where the online policy competes versus adversarial resource demands. The performance metric is the regret, i.e. the cost difference between our policy and the best static reservation with knowledge of the entire demand sample path. We seek to find an online policy that achieves zero average regret under its worst adversary (a condition known as "no regret"), while we require from our policy to satisfy a time-average constraint that concerns the number of violations occurring in the studied time period. To the best of our knowledge, no previous work has addressed the problem of reserving cloud resources in this setting. The main contribution of this paper is to design the Time Horizon Online Reservation (THOR): a feasible "no regret" resource reservation policy.

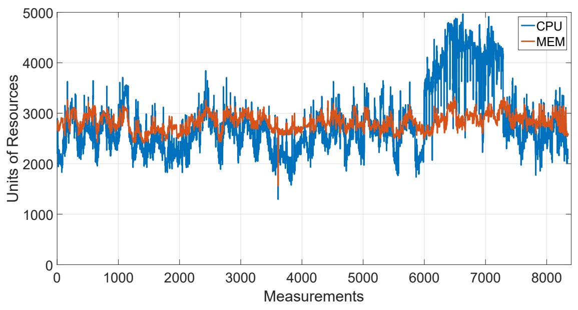


Fig. 1. Aggregate resource utilization of the Google Cluster. The resources are normalized with respect to the server with the highest memory and CPU. Every point corresponds to 5mins, up to 29 days measured. The fluctuations are characterized as unpredictable in [6].

A. Prior Work

The framework of OCO is used to minimize the sum of convex functions $\sum_t f_t(x_t)$ where f_t is revealed to the optimizer after the action x_t is taken. It was inspired by the seminal paper of Zinkevich [2], who proposed to predict f_t as a linearization of f_{t-1} and take a gradient step in the direction of $\nabla f_{t-1}(x_{t-1})$. OCO allows us to design model-free algorithms that are data-driven and robust to environment changes cf. [7], [8], since the "no regret" property is extremely powerful; it implies that our online algorithm learns to allow the same average losses as a static policy that knows the future. We study a slightly perturbed setting of OCO, where the function f is static and known, but the constraint function $\sum_t g_t(x_t) \leq 0$ is chosen by the adversary.

A number of past works have focused on constrained-OCO problems. The simplest case is when the constraint is not adversarial, and limits the policy actions in the same manner at each time slot. This is addressed in the online gradient of Zinkevich via a projection, but to extend to cases with complicated sets [9], [10] proposed an alternative approach based on Lagrangian relaxation. If we have a time-average constraint coupling the decisions over time, [11] uses self-concordant barrier functions to relax it, while [9], [10] provides a dual algorithm. While these methods ensure asymptotic feasibility–the constraint residual $\sum_t g(x_t)$ scales as $o(T)$–they do not apply to our problem, where the constraint set is shaped by adversary-selected (time changing) convex functions.

The well-known result of [12] states that in general it is impossible to simultaneously achieve $o(T)$ regret and asymptotic feasibility in constrained-OCOs where both objective and constraint set are tinkered by an adversary. However, [13] showed that it is possible when there exists a static solution which strictly satisfies all constraint functions at every slot (a Slater vector). With the approach of [13], however, the "no regret" property is provided with respect to the Slater vector, meaning that applying [13] to our problem will yield a feasible resevation policy, but with a poor cost guarantee due to the restrictive assumption of ensuring the constraint at every slot. Consider a benchmark static reservation that only satisfies the average constraint every K slots. Then as K increases, the constraint is looser and we obtain a stronger guarantee, but establishing the guarantee may become harder. While [13] proves the case $K = 1$, in this paper we prove the case where $K = O(T^{1-\varepsilon})$, and propose the online policy THOR, which is asymptotically feasible and provides $o(T)$ regret.

B. Our Contribution

We formulate the problem of reserving resources for cloud computing as a constrained-OCO. At each slot, *(i)* an online reservation policy decides a reservation vector, then *(ii)* the adversary decides a demand vector, and last *(iii)* a cost is paid for the reserved resources and a violation is noted if the demand was not covered by the reservation. A reservation policy is feasible if at the end of the T-slot horizon the number of violations of resource i are no more than a configurable $\epsilon_i T$. *We seek to find a feasible policy that achieves no regret with respect to the best static reservation in hindsight while satisfying the violation constraint at all K-slot windows within* T. Our contributions are summarized as follows:

- We introduce a natural machine learning approach for reserving resources for cloud computing. Our constrained-OCO framework is an ideal setup for investigating more complicated scenarios with reservations, e.g., reserving resource slices in wireless networks.
- We propose THOR, a policy we prove to achieve asymptotic feasibility and "no regret" with respect to a benchmark constrained to $K = O(T^{1-\varepsilon})$ slots, the first of its kind. The performance guarantees of THOR are obtained by a novel combination of the Lyapunov K-slot drift technique with the linearization idea of Zinkevich. THOR inherits the simplicity of online gradient, and therefore is straightforward to implement in practical systems.
- We have validated THOR resource reservations using a public dataset provided by Google [14]. THOR vastly outperforms our implementation of the textbook Follow The Leader (FTL) policy in guaranteeing the violations constraint, while it achieves similar or sometimes better performance than the static oracle T-slot policy, in the challenging, non-stationary CPU workload.

II. System Model

Requests. Our system operates in slots $t = 1, \ldots, T$, with T being the horizon. In slot t the cloud users request λ_i^t units of resource i (for example $i = 1$ refers to CPU and $i = 2$ to memory). We consider I types of resources. To model the fact that the vectors $\boldsymbol{\lambda}^t$ are drawn from a general distribution $D(\boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^T)$ (hence model-free), we allow them to be selected by an adversary who aims to harm our system.

Reservations. A reservation policy π decides at each slot to reserve $x_i^{t,\pi}$ units of resource i. Formally, at time t an *online reservation policy* is a mapping from past requests to a vector of nonnegative values:

$$\pi : (\boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^{t-1}, \boldsymbol{x}^1, \ldots, \boldsymbol{x}^{t-1}) \to \mathbb{R}_+^I.$$

The above is depictive of the action order within a slot, i.e., first a reservation $\boldsymbol{x}^{t,\pi}$ is made, and then the adversary reveals the values $\boldsymbol{\lambda}^t$. There is a cost c_i attached to each resource, and therefore at the end of slot t, policy π incurs a cost

$$C(\boldsymbol{x}^{t,\pi}) = \sum_{i=1}^{I} c_i x_i^{t,\pi}.$$

Violation guarantees. Let v_i^t denote the event of resource i violation in slot t, which occurs when the request for a resource exceeds the reservation, i.e., $v_i^t \triangleq \mathbb{1}\left\{\lambda_i^t > x_i^{t,\pi}\right\}$. A policy π is called *feasible* if the time-average violations of resource i do not exceed a pre-determined threshold ϵ_i:

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[v_i^t\right] \leq \epsilon_i, \quad \text{for all} \;\; i = 1, \ldots, I.$$

Hence, the feasibility constraint of resource i can also be written in the form:

$$\sum_{t=1}^{T} \mathbb{P}\left(\lambda_i^t > x_i^t\right) \leq \epsilon_i T. \tag{1}$$

Observe that the above constraint couples the decisions across the entire horizon. Our initial objective is to find a *feasible* policy that minimizes the total cost $\sum_{t=1}^{T} C(\boldsymbol{x}^{t,\pi})$, however, since in slot t the arrivals $\boldsymbol{\lambda}^t$ are unknown, such an objective is out of reach. Next, we provide an alternative approach through the framework of *Online Convex Optimization* (OCO).

Regret. We introduce the performance metric of *regret*, which is commonly used in the literature of machine learning to measure the robustness of online algorithms [7], [8]. The regret R_T^π is the cumulative difference of losses between policy π and a benchmark policy which is aware of the entire sample path $\boldsymbol{\lambda}^1, \dots, \boldsymbol{\lambda}^T$ but forced to take a static action throughout the horizon–often called *best static policy in hindsight*. Specifically, let $\boldsymbol{x}^*$ denote our benchmark, which is calculated as the solution to the following problem:

$$\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}_+^I} T\,C(\boldsymbol{x}) \quad \text{s.t.} \quad \sum_{t=1}^{T} \mathbb{P}\left(\lambda_i^t > x_i^*\right) \le \epsilon_i T.$$

Then the regret is defined as follows:

$$R_T^\pi = \inf_{D}\ \mathbb{E}\left[\sum_{t=1}^{T} C(\boldsymbol{x}^{t,\pi}) - T\,C(\boldsymbol{x}^*)\right],$$

where the infimum is taken w.r.t. the supported distributions of the adversary, and the expectation w.r.t. the (possibly) randomized $\boldsymbol{x}^{t,\pi}, \boldsymbol{\lambda}^t$. If $R_T^\pi = o(T)$, then we say that policy π has "no regret", since $R_T^\pi/T \to 0$ as $T \to \infty$, i.e., the average losses from the benchmark are amortized. Our goal is to obtain a *feasible* online reservation policy with "no regret".

A. Modified Adversary

In this subsection we introduce two innovations with respect to the classical OCO framework, one related to the decisions of the adversary, and one to the benchmark we compare against.

Probabilistic adversary support. It is customary to limit the actions of the adversary to be no more than a finite value $\Lambda_{i,\max}$ for resource i. In our problem, this hard constraint is problematic: *(i)* small $\Lambda_{i,\max}$ (e.g. set equal to the maximum observed value) will cause our algorithms to "think" that an action $x_i^t = \Lambda_{i,\max}$ ensures a violation-free slot, which will incur instability should a flow of larger values occurs in the future, while *(ii)* large $\Lambda_{i,\max}$ will force our algorithms to consistently overbook in order to ensure no violation, leading to very poor performance.

To address this issue, we propose the idea of bounding the adversary with a stationary process with known distribution; the distribution is configured based on the data. Specifically, in this paper we set Λ_i^t to be an i.i.d. Gaussian process, and then restrict the adversary to $\lambda_i^t \in [0, \Lambda_i^t]$. The benefit of this approach lies in the elasticity offered by the stationary process. Due to the concavity of its cumulative distribution, our algorithms will be able to learn tradeoffs between the probability of violations and the investment cost. We mention that despite $\boldsymbol{\Lambda}^t$ being stationary, the actual arrivals $\boldsymbol{\lambda}^t$ remain model-free and possibly non-stationary.

K-slot feasibility. When showing that policy π has no regret, we are effectively showing that π achieves the same average performance with the benchmark. It is useful then to introduce a class of benchmark policies that ensure the

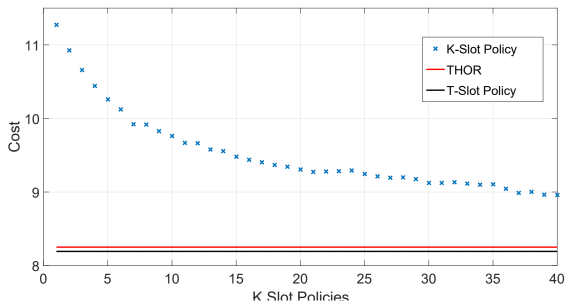


Fig. 2. Average cost comparison in a repeated randomly generated instance. Adversary uniformly selects $\lambda_i^t \in [0, \Lambda_i^t]$ in the course of the experiment, while Λ_i^t is maintained between the multiple runs of the experiment. The plotted points represent the cost of K(=[1,40])-slot benchmark policy, the black line is the T-slot benchmark policy and the red line is our policy.

violation constraint for all windows of K slots within the horizon T:

$$\boldsymbol{x}^*(K) \in \arg\min_{\boldsymbol{x}\in\mathbb{R}_+^I} T\,C(\boldsymbol{x}) \tag{2}$$

$$\text{s.t.} \sum_{k=0}^{K-1} \mathbb{P}\left(\lambda_i^{t+k} > x_i^*\right) \le \epsilon_i K, \quad \forall t = 1, \dots, T-K.$$

Observe that for $K = T$ we obtain the original benchmark. However, as K decreases, we will have an interesting trade off: on one hand, the benchmark should ensure the average violations in shorter periods, hence it will incur higher investment costs (as the optimization above will have a stricly smaller constraint set), on the other hand it might be easier to prove the "no regret" property. Indeed, prior work [13] produced a "no regret" policy versus a benchmark which satisfies the violation constraint at every slot ($K = 1$). Note, however, that such a guarantee is compromised in our problem. For example, in Fig. 2 we plot the performance of K benchmarks, where we observe greatly increased cost for $K = 1$. We also observe that, initially increasing K enhances greatly the achieved performance, but the returns are diminishing. In this paper, we will obtain a "no regret" guarantee for the case when $K = O(T^{1-\varepsilon})$.

III. Queue Assisted Online Learning Algorithm

In this section we present our algorithm, and provide intuition into its functionality. An online reservation policy is called in slot t to update the decisions from $\boldsymbol{x}^{t-1}$ to $\boldsymbol{x}^t$. In order to explain how THOR performs this update, we will discuss some intermediate steps, namely *(i)* the constraint convexification *(ii)* the predictor queue, and *(iii)* the drift plus penalty plus smoothness. Finally, we will present THOR and show how it naturally arises from these three steps. Formal performance guarantees are presented in the following section.

A. Constraint Convexification

In this subsection, we propose a convex approximation for the feasibility constraint Eq.(1). This constraint will be tighter as it will become obvious below. We presented in the system model $\mathbb{P}\left(\lambda_i^t > x_i^t\right)$ to be the quantile function of the adversary for every slot t and resource i. This function is a non-increasing function, but can be non-convex. For the approximation we will use $\mathbf{\Lambda}$ which we assume to be an i.i.d. Gaussian process that caps the distribution of the adversary $\lambda_i^t \leq \Lambda_i^t$. For ease of exposition, we define $F_{\lambda_i^t}(x_i^t) \triangleq \mathbb{P}\left(\lambda_i^t > x_i^t\right)$. By our assumption, it is true that:

$$F_{\lambda_i^t}(x_i^t) \leq F_{\Lambda_i^t}(x_i^t).$$

Furthermore, since the quantile function of the Gaussian distribution is quasiconvex we can design a convex envelope function for F_Λ:

$$F_{E_i^t}(x_i^t) = \begin{cases} G_i^t\, x_i^t + \beta_i, \ x^t < \mu_i \\ F_{\Lambda_i^t}(x_i^t), \ \text{otherwise}, \end{cases}$$

where $G_i^t = F'_{\Lambda_i^t}(\mu_i)$ and $F_{\Lambda_i^t}(x_i^t) \leq F_{E_i^t}(x_i^t)$. Hence, for reservations less than μ_i the CCDF is enveloped by a linear function that decreases by G_i^t. We note that this is defined for the theoretical proofs to follow through, but also it is relevant to practice, in the unlikely case of loose guarantees (big ϵ_i) or extremely optimistic error predictions. This envelope will produce strong derivatives to increase the reservations (while the gaussian CCDF will have a smaller -but still negative-slope). Hereinafter, we simplify the notation to $F_t(x_i^t)$, to describe $F_{E_i^t}(x_i^t)$. We further note that, $0 \leq F_{E_i^t}(x_i^t) \leq F$, where F is a constant and that there exists a constant $G \geq G_i^t$.

B. Predictor Queue

Intuitively, we could use $F_t(x_i^t)$ to take a good step in slot t towards ensuring the time-average constraint, but this information is not available in our model: the function F_t relates to the choice of the adversary that takes place after we commit our decision $\boldsymbol{x}_t$. What is available is the previous value $F_{t-1}(x_i^{t-1})$. Inspired by the work of Zinkevich [2], we introduce a linear *prediction of the violation probability* $F_t(x_i^t)$:

$$b_i(x_i^t) \triangleq F_{t-1}(x_i^{t-1}) + F'_{t-1}(x_i^{t-1})(x_i^t - x_i^{t-1}), \qquad (3)$$

where $F'_{t-1}(x_i^{t-1})$ denotes the derivative, hence $b_i(x_i^t)$ is the first order Taylor expansion of F_{t-1} around x_i^{t-1} evaluated at x_i^t. Note that only x_i^t in Eq.(3) is to be determined at time t.

Definition 1 (Predictor Queue Vector). *We define as $\boldsymbol{Q}$ the Predictor Queue Vector. Every element of the vector is a virtual queue for each resource, containing the sum of the predicted probabilities of violation in the past iterations.*

$$Q_i(t+1) = [Q_i(t) - \epsilon_i + b_i(x_i^t)]^+. \qquad (4)$$

Virtual queue $Q_i(t)$ is a counter that increases with our controllable predictions $b_i(x_i^t)$ and decreases at a steady rate ϵ_i. If we limit the growth of $Q_i(t)$ to $o(T)$, then the average (predicted) violations would only overshoot ϵ_i by an amortizable amount, which can be manipulated into providing asymptotic feasibility. In fact, we will rigorously prove this intuition in section IV-B.

C. Drift Plus Penalty Plus Smoothness for Online Learning

Having obtained a handle on the time-average constraint (and hence also asymptotic feasibility) via the predictor queue, it remains to explain how $\boldsymbol{x}_t$ is updated in THOR. To combine the consideration of the cost and the predictor queue we will use the technique of Drift Plus Penalty (DPP), a framework used to solve constrained stochastic network optimization problems [1]. In DPP, the tradeoff between the constraints (queue lengths) and the cost is controlled via the penalty parameter V. Specifically, first we consider the quadratic Lyapunov drift (defined as $\Delta(t) = \frac{1}{2}\sum_i Q_i(t+1)^2 - \frac{1}{2}\sum_i Q_i(t)^2$), which measures the impact of our policy on the norm of the predictor queue vector. The drift can be bounded as in Lemma 4.2 in [15],

$$\Delta(t) \leq B + \sum_{i=1}^{I} Q_i(t)[b_i(x_i^t) - \epsilon_i],$$

where $B = \frac{1}{2}I(\max\{GD, F\})^2$ is a constant. Then, adding to both parts of the drift inequality the *penalty* term, defined as the cost function multiplied by a weight V, we arrive at the drift plus penalty inequality:

$$\Delta(t) + VC(\boldsymbol{x}) \leq B + \sum_{i=1}^{I} Q_i(t)[b_i(x_i^t) - \epsilon_i] + VC(\boldsymbol{x}). \qquad (5)$$

A large $\boldsymbol{x}$ tends to minimize the drift term $\Delta(t)$ but incurs a high cost $VC(\boldsymbol{x})$, while small $\boldsymbol{x}$ has the opposite effect. Clearly, minimizing DPP achieves a balance between the two conflicting objectives. Remarkably, prior work in DPP shows that finding the minimizer of the upperbound in Eq.(5) at every slot, eventually produces an online policy that simultaneously achieves a cost within $O(\frac{1}{V})$ of the optimal and bounds the queue with $O(V)$.

Here, we will further add a quadratic (Tikhonov) regularizer [16] centered at the previous iterate $\boldsymbol{x}^{t-1}$, this will encourage the new reservation $\boldsymbol{x}^t$ to not drastically change its value compared to the last iteration $\boldsymbol{x}^{t-1}$.

Definition 2 (Drift Plus Penalty Plus Smoothness). *By adding the penalty term $\alpha||\boldsymbol{x}^t - \boldsymbol{x}^{t-1}||^2$ on both sides of the inequality Eq.*(5) *we get the upper bound on Drift Plus Penalty Plus Smoothness (DPPPS).*

$$\begin{aligned} &\Delta(t) + VC(\boldsymbol{x}^t) + \alpha||\boldsymbol{x}^t - \boldsymbol{x}^{t-1}||^2 \leq \qquad (6) \\ &\quad B + \sum_{i=1}^{I} Q_i(t)[b_i(x_i^t) - \epsilon_i] + VC(\boldsymbol{x}^t) + \alpha||\boldsymbol{x}^t - \boldsymbol{x}^{t-1}||^2. \end{aligned}$$

also we define the upper bound as a function of $\boldsymbol{x}$:

$$g(\boldsymbol{x}) \triangleq B + \sum_{i=1}^{I} Q_i(t)[b_i(x_i^t) - \epsilon_i] + VC(\boldsymbol{x}) + \alpha||\boldsymbol{x} - \boldsymbol{x}^{t-1}||^2.$$

to be used in the theoretical analysis later.

Asymptotically the effect of the regularizer fades, so the optimality results are unaffected. While, however, the original DPP yields online policies that constantly operate at two extremes (called bang-bang policies), with this regularizer THOR update is transformed to a smooth gradient step, as we show next.

D. Online Reservation Policy

Proposition 1 (THOR minimizes DPPPS bound)**.** *The updates*

$$x_i^t = \left[x_i^{t-1} - \frac{1}{2\alpha}(Vc_i + Q_i(t)F'_{t-1}(x_i^{t-1}))\right]^+. \quad (7)$$

minimize at each slot t the upper bound on the predicted DPPPS Eq.(6).

Proof. The function $g(\boldsymbol{x})$, found in Def. 2, is decomposable to the I resources and the minimizer of each component is:

$$\begin{aligned} x_i^t &= \underset{x_i \geq 0}{\operatorname{argmin}}\{g(\boldsymbol{x})\} \\ &= \underset{x_i \geq 0}{\operatorname{argmin}}\{Q_i(t)F'_{t-1}(x_i^{t-1})x_i + Vc_ix_i + \alpha(x_i - x_i^{t-1})^2\}. \end{aligned}$$

From this expression, it becomes apparent that, by removing the regularizer ($\alpha = 0$), since F is a quantile function and F' is negative, we get the following:

$$x_i^t = \begin{cases} 0, \text{ if } Vc_i \geq |Q_i(t)F'_{t-1}(x_i^{t-1})| \\ +\infty, \text{ otherwise.} \end{cases}$$

This generates a bang-bang reservation policy for every time slot t. Bang-bang policies are ideal for scheduling or routing, but are not practical for a cloud resource reservation environment. Here, we must maintain as stable reservations as possible, allowing slow installation or removal of servers and resources. Meanwhile, with the added regularizer, the reservation update is a gradient minimization step with step size $\frac{1}{2\alpha}$. This comes naturally by finding the stationary point:

$$\begin{aligned} Q_i(t)F'_{t-1}(x_i^{t-1}) + Vc_i + 2\alpha x_i - 2\alpha x_i^{t-1} = 0 \\ x_i = \left[x_i^{t-1} - \frac{1}{2\alpha}(Vc_i + Q_i(t)F'_{t-1}(x_i^{t-1}))\right]^+. \end{aligned}$$

□

In conclusion the evolution of THOR policy can be described as follows:

Time Horizon Online Reservations (THOR)

Initializition: Predictor queue initial length $\boldsymbol{Q}(1) = \boldsymbol{0}$, initial reservation vector $\boldsymbol{x}^0 \in \mathbb{R}_+^I$.
Parameters: penalty constant V, step size α, cost of resource unit c_i, constraint requirement per resource $\epsilon_i \leq 0.5$.
Updates at every time slot $t \in \{1, \dots, T\}$**:**

$$x_i^t = \left[x_i^{t-1} - \tfrac{1}{2\alpha}(Vc_i + Q_i(t)F'_{t-1}(x_i^{t-1}))\right]^+, \quad (8)$$
$$Q_i(t+1) = [Q_i(t) + b_i(x_i^t) - \epsilon_i]^+. \quad (9)$$

Where $b_i(x_i^t)$ is given in Eq.(3) and $F'_t(x_i^t)$ is the derivative of the convexified CCDF $F_{E_i^t}$.

In the following section we will use the above-explained intuition to establish rigorous proofs that our THOR algorithm is asymptotically feasible and has "no regret" against the K benchmark. These results are harder to achieve than those from the standard DPP framework, because we have no knowledge of the current status of the queue ($F_t(x_i^t)$) and also the average violations (arrivals to the virtual queue) are arbitrarily distributed (selected by a constrained adversary), hence there are no Markovian statistics to be learned.

IV. Performance Analysis

In this section we prove that THOR is a feasible policy with no regret against K-slot policies. First we will show a general upper bound on THOR DPPPS by using the strong convexity property of $g(\boldsymbol{x})$. Then we will use this general bound to achieve a sublinear in time bound on the predictor queues of THOR, which will be used to prove feasibility, i.e. no time average constraint violation in a time horizon T. Finally, we compare THOR with a K benchmark, using the K benchmark properties, to prove the THOR's no regret. The results are summarized in a corollary at the end of the section.

We note that since $\boldsymbol{x}^t$ by Eq.(7) is the minimizer of the upper bound on DPPPS (Eq.(6)) for every time slot t; any static reservation $\boldsymbol{y} \in \mathbb{R}_+^I$ bounds THOR's DPPPS by above. This is an important aspect for the analysis to follow.

$$\begin{aligned} B + \sum_{i=1}^I Q_i(t)[b_i(x_i^t) - \epsilon_i] + VC(\boldsymbol{x}^t) + \alpha||\boldsymbol{x}^t - \boldsymbol{x}^{t-1}||^2 \leq \\ B + \sum_{i=1}^I Q_i(t)[b_i(y_i) - \epsilon_i] + VC(\boldsymbol{y}) + \alpha||\boldsymbol{y} - \boldsymbol{x}^{t-1}||^2. \end{aligned}$$

A stronger condition is required in our analysis, a more refined upper bound which is achieved due to the imposed strong convexity of the Tikhonov regularizer. The following lemma is a key lemma for our results.

Lemma 1. *[Strong Convexity Bound] Let $\boldsymbol{y} \in \mathbb{R}_+^I$ be a static reservation, then the DPPPS of THOR $\boldsymbol{x}^t$ is bounded by:*

$$\begin{aligned} \Delta(t) + VC(\boldsymbol{x}^t) + \alpha||\boldsymbol{x}^t - \boldsymbol{x}^{t-1}||^2 \leq B + VC(\boldsymbol{y}) + \\ \sum_{i=1}^I Q_i(t)[F_t(y_i) - \epsilon_i] + \alpha||\boldsymbol{y} - \boldsymbol{x}^{t-1}||^2 - \alpha||\boldsymbol{y} - \boldsymbol{x}^t||^2. \end{aligned}$$

Proof. Due to $g(\boldsymbol{x})$ being 2α-strongly convex, we have $g(\boldsymbol{x}_{min}) = g(\boldsymbol{y}) - \frac{2\alpha}{2}||\boldsymbol{y} - \boldsymbol{x}_{min}||^2$, for all $\boldsymbol{y} \in \mathbb{R}_+^I$ static policies. This refines THOR's upper bound:

$$\begin{aligned} &\Delta(t) + VC(\boldsymbol{x}^t) + \alpha||\boldsymbol{x}^t - \boldsymbol{x}^{t-1}||^2 \leq \\ &B + \sum_{i=1}^I Q_i(t)[b_i(x_i^t) - \epsilon_i] + VC(\boldsymbol{x}^t) + \alpha||\boldsymbol{x} - \boldsymbol{x}^{t-1}||^2 \leq \\ &B + \sum_{i=1}^I Q_i(t)[b_i(y_i) - \epsilon_i] + VC(\boldsymbol{y}) + \\ &\qquad \alpha||\boldsymbol{y} - \boldsymbol{x}^{t-1}||^2 - \alpha||\boldsymbol{y} - \boldsymbol{x}^t||^2 \overset{(1)}{\leq} \\ &B + \sum_{i=1}^I Q_i(t)[F_t(y_i) - \epsilon_i] + VC(\boldsymbol{y}) + \\ &\qquad \alpha||\boldsymbol{y} - \boldsymbol{x}^{t-1}||^2 - \alpha||\boldsymbol{y} - \boldsymbol{x}^t||^2. \end{aligned}$$

For (1) we use the convexity of the envelope CCDF function $F_{E_i^t}$, $F_t(y_i) \geq b_i(y_i)$. □

A. Predictor Queue Upper Bound

The first important step is to prove that under our policy Eq.(8)-(9) the predictor virtual queues increase sublinearly with time. This will later give us a bound for the constraint violation which ensures THOR policy's feasibility.

Lemma 2 (Upper Bound on Predictor Queues)**.** *Let D be a finite upper bound on the maximum reservation, such that $x_i \leq x_{max} = D$. The queue predictor vector $\boldsymbol{Q}$ at time slot T satisfies the following inequalities:*

$$||Q(T+1)||_2 \leq \sqrt{2BT + 2VT\sum_{i=1}^{I} c_i r_i(\epsilon_i) + 2\alpha I D^2},$$

$$||Q(T+1)||_1 \leq \sqrt{I}||Q(T+1)||_2,$$

where $r_i(\epsilon_i) \triangleq \mu_i + \sigma_i \mathcal{Q}^{-1}(\epsilon_i)$.

Proof. We will use Lemma 1, to prove that a static reservation $\bar{\boldsymbol{y}}$ achieves sublinear growth in T of the $\boldsymbol{Q}$ vector. Then, since our policy x_i^t is the minimizer of the DPPPS, our policy achieves (at least) the same upper bound. Select $\bar{\boldsymbol{y}}$ to be a static reservation that always satisfies the constraint $F_{\Lambda_i^t}(\bar{y}_i) \leq \epsilon_i,\ \forall i, t$. It is easy to show that $\bar{y}_i \geq \mu_i + \sigma_i \mathcal{Q}^{-1}(\epsilon_i)$, where $\mathcal{Q}$ is the quantile function of the standard normal distribution. We take $\bar{y}_i = \mu_i + \sigma_i \mathcal{Q}^{-1}(\epsilon_i)$, which gives the cost:

$$C(\bar{\boldsymbol{y}}) = \sum_{i=1}^{I} c_i(\mu_i + \sigma_i \mathcal{Q}^{-1}(\epsilon_i)) = \sum_{i=1}^{I} c_i r_i(\epsilon_i).$$

According to Lem.1:

$$\Delta(t) + \underbrace{VC(\boldsymbol{x}^t) + \alpha||\boldsymbol{x}^t - \boldsymbol{x}^{t-1}||^2}_{(a)} \leq B + VC(\bar{\boldsymbol{y}}) +$$
$$\underbrace{\sum_{i=1}^{I} Q_i(t)[F_{E_i^t}(\bar{y}_i) - \epsilon_i]}_{(b)} + \alpha||\bar{\boldsymbol{y}} - \boldsymbol{x}^{t-1}||^2 - \alpha||\bar{\boldsymbol{y}} - \boldsymbol{x}^t||^2.$$

Here, terms (a) and (b) can be discarded. (a) is positive, while (b) is negative, due to $F_{E_i^t}(\bar{y}_i) \leq \epsilon_i, \forall i, t$. This leaves us with:

$$\Delta(t) \leq B + VC(\bar{\boldsymbol{y}}) + \alpha||\bar{\boldsymbol{y}} - \boldsymbol{x}^{t-1}||^2 - \alpha||\bar{\boldsymbol{y}} - \boldsymbol{x}^t||^2.$$

We take the telescopic sum over the time slots $\{1, ..., T\}$:

$$\sum_{t=1}^{T} \Delta(t) \leq BT + VC(\bar{\boldsymbol{y}})T +$$
$$\alpha\sum_{t=1}^{T} ||\bar{\boldsymbol{y}} - \boldsymbol{x}^{t-1}||^2 - \alpha\sum_{t=1}^{T} ||\bar{\boldsymbol{y}} - \boldsymbol{x}^t||^2.$$

By taking the telescopic sum the intermediate terms of (i) the quadratic Lyapunov drift $\Delta(t)$ and (ii) the norms cancel each other. Furthermore, $Q(1) = 0$ and the (negative) norm term $-\alpha||\bar{\boldsymbol{y}} - \boldsymbol{x}^T||^2$ can be dropped. We replace $VC(\bar{\boldsymbol{y}})$ with its cost and we arrive at:

$$\frac{1}{2}\sum_{i=1}^{I} Q_i(T+1)^2 \leq VT\sum_{i=1}^{I} c_i r_i(\epsilon_i) + BT + \alpha||\bar{\boldsymbol{y}} - \boldsymbol{x}^0||^2.$$

Since $x_i \leq x_{max} = D$, then $||\boldsymbol{x} - \boldsymbol{y}|| \leq \sqrt{I}D,\ \forall \boldsymbol{y}, \boldsymbol{x} \in \mathbb{R}_+^I$.

$$\sum_{i=1}^{I} Q_i(T+1)^2 \leq 2BT + 2VT\sum_{i=1}^{I} c_i r_i(\epsilon_i) + 2\alpha I D^2$$

$$||Q(T+1)||_2 \leq \sqrt{2BT + 2VT\sum_{i=1}^{I} c_i r_i(\epsilon_i) + 2\alpha I D^2}.$$

The result for the bound on the 1-norm follows by using the norm inequality $||x||_1 \leq \sqrt{I}||x||_2$, for I vector elements. □

B. Constraint Residual

Having established a bound on $\boldsymbol{Q}$, our next objective is to use it to bound the constraint violations at time slot T and obtain asymptotic feasibility. For simplicity, hereon we denote

$$Q^B(T+1) \triangleq \sqrt{2BT + 2VT\sum_{i=1}^{I} c_i r_i(\epsilon_i) + 2\alpha I D^2}.$$

Theorem 1 (Upper Bound on Constraint Violation)**.** *The constraint violation for each resource is bounded by the size of the predictor queue at time $T+1$, $Q_i(T+1)$, which is upper bounded by $Q^B(T+1)$:*

$$\sum_{t=0}^{T-1} F_t(x_i^t) - \epsilon_i T \leq Q^B(T+1) + \frac{G^2\sqrt{2B}T(T+1)}{4\alpha},$$

where $G \geq |G_i^t|$ is the upper bound of the absolute value of the derivative of the quantile function F defined in Sect.III-A.

Proof. We take the predictor queue update equation Eq.(4):

$$Q_i(t+1) = [Q_i(t) + b_i(x_i^t) - \epsilon_i]^+ \geq Q_i(t) + b_i(x_i^t) - \epsilon_i$$
$$\geq Q_i(t) + F_{t-1}(x_i^{t-1}) - G(x_i^t - x_i^{t-1}) - \epsilon_i,$$

We have by projection properties and by the queue update policy Eq.(7) that $||x_i^t - x_i^{t-1}||_1 \leq \frac{||Vc_i - Q_i(t)G||_1}{2\alpha} \leq \frac{GQ_i(t)}{2\alpha}$:

$$F_{t-1}(x_i^{t-1}) - \epsilon_i \leq Q_i(t+1) - Q_i(t) + \frac{G^2 Q_i(t)}{2\alpha}.$$

We sum for the time slots $t = \{1, ..., T\}$:

$$\sum_{t=1}^{T} F_{t-1}(x_i^{t-1}) - \epsilon_i T \leq Q_i(T+1) + \frac{G^2\sum_{t=1}^{T} Q_i(t)}{2\alpha}.$$

We have $Q_i(t+1) \leq Q_i(t) + ||b_i(x_i^t) - \epsilon_i||_1 \leq Q_i(t) + \sqrt{2B}$, hence $Q(t) \leq t\sqrt{2B},\ \forall t$ and $Q_i(T+1) \leq Q^B(T+1)$. Using those on the above equation we get:

$$\sum_{t=0}^{T-1} F_t(x_i^t) - \epsilon_i T \leq Q^B(T+1) + \frac{G^2\sum_{t=1}^{T} t\sqrt{2B}}{2\alpha}.$$

The result follows. □

By properly selecting the constants V, α and for growing T the residual of the constraint on average goes to zero. In the next subsection, we will show that our policy guarantees similar cost against the K-Slot best static policy.

C. No Regret Against the K Benchmark Policy

Here, we will prove that our policy achieves no regret against K-slot benchmark policy, these are defined in Sect. II by Eq.(2). We start by the upper bound on DPPPS by Lem.(1), we prove a small technical lemma to be able to use the properties of the K slot benchmark; enabling us to upper bound THOR's performance in comparison to the benchmark.

Theorem 2 (Upper Bound Against K Benchmark)**.** *Regret, against the optimal static policy in $\mathbb{R}_+^I$ that achieves feasibility in K slots, is upper bounded by:*

$$\sum_{t=1}^{T} C(\boldsymbol{x}^t) - \sum_{t=1}^{T} C(\boldsymbol{x}^\star) \leq \frac{BKT}{V} + \frac{\alpha I D^2}{V} + \frac{IB(K+1)(2K+1)}{6V} + IDc_i(K-1).$$

Proof. We begin by Lem.(1), by setting $\boldsymbol{y} = \boldsymbol{x}^*(K)$ and summing the inequality for K slots, $t = \{1, 2, \dots, K\}$:

$$\sum_{\tau=0}^{K-1} \left[\Delta(t+\tau) + VC(\boldsymbol{x}^{t+\tau}) + \underbrace{\alpha ||\boldsymbol{x}^{t+\tau} - \boldsymbol{x}^{t+\tau-1}||^2}_{(a)} \right] \leq$$
$$BK + V \sum_{\tau=0}^{K-1} C(\boldsymbol{y}) + \underbrace{\sum_{\tau=0}^{K-1} \sum_{i=1}^{I} Q_i(t+\tau)[F_{t+\tau}(y_i) - \epsilon_i]}_{(b)} +$$
$$\alpha \sum_{\tau=0}^{K-1} ||\boldsymbol{y} - \boldsymbol{x}^{t+\tau-1}||^2 - \alpha \sum_{\tau=0}^{K-1} ||\boldsymbol{y} - \boldsymbol{x}^{t+\tau}||^2. \tag{10}$$

Lemma 3. *For policy $\boldsymbol{y} = \boldsymbol{x}^*(K)$, for any $t \in \{1, \dots, T\}$:*

$$\sum_{\tau=0}^{K-1} \sum_{i=1}^{I} Q_i(t+\tau)[F_t(y_i) - \epsilon_i] \leq IBK(K-1).$$

Proof. We give a lower bound for $Q(t+K)$ for any $K \geq 1$:

$$Q_i(t+K) \geq Q_i(t) + \sum_{\tau=0}^{K-1} [F_{t+\tau}(y_i) - \epsilon_i], \tag{11}$$

and an upper bound:

$$Q_i(t+K) \leq Q_i(t) + \sum_{\tau=0}^{K-1} ||F_{t+\tau}(y_i) - \epsilon_i||_1. \tag{12}$$

Next, we use these bounds of the queue derived above, so that $Q_i(t)$ appears as a common term in the sum. We will prove for a single Queue $q_i(t)$. We denote $\max\{0, f(\cdot)\} \triangleq [f(\cdot)]^+$ and $\min\{0, f(\cdot)\} \triangleq [f(\cdot)]^-$:

$$\sum_{\tau=0}^{K-1} Q_i(t+\tau)[F_{t+\tau}(y_i) - \epsilon_i] =$$
$$\sum_{\tau=0}^{K-1} Q_i(t+\tau) \left\{ [F_{t+\tau}(y_i) - \epsilon_i]^+ + [F_{t+\tau}(y_i) - \epsilon_i]^- \right\} \overset{(a)}{\leq}$$
$$\sum_{\tau=0}^{K-1} \left\{ Q_i(t) + \sum_{j=0}^{\tau-1} ||F_{t+j}(y_i) - \epsilon_i||_1 \right\} [F_{t+\tau}(y_i) - \epsilon_i]^+ +$$
$$\sum_{\tau=0}^{K-1} \left\{ Q_i(t) + \sum_{j=0}^{\tau-1} [F_{t+j}(y_i) - \epsilon_i] \right\} [F_{t+\tau}(y_i) - \epsilon_i]^- \overset{(b)}{\leq}$$
$$Q_i(t) \sum_{\tau=0}^{K-1} \{ [F_{t+\tau}(y_i) - \epsilon_i]^+ + [F_{t+\tau}(y_i) - \epsilon_i]^- \} +$$
$$\sum_{\tau=0}^{K-1} \left\{ \sum_{j=0}^{\tau-1} ||F_{t+j}(y_i) - \epsilon_i||_1 \right\} [F_{t+\tau}(y_i) - \epsilon_i)]^+ +$$
$$\sum_{\tau=0}^{K-1} \left\{ \sum_{j=0}^{\tau-1} [F_{t+j}(y_i) - \epsilon_i] \right\} [F_{t+\tau}(y_i) - \epsilon_i]^- \overset{(c)}{\leq}$$
$$Q_i(t) \sum_{\tau=0}^{K-1} [F_{t+\tau}(y_i) - \epsilon_i] +$$
$$\sum_{\tau=0}^{K-1} \left\{ \sum_{j=0}^{\tau-1} ||F_{j+\tau}(y_i) - \epsilon_i||_1 \right\} ||F_{t+\tau}(y_i) - \epsilon_i||_1 \overset{(d)}{\leq}$$
$$Q_i(t) \sum_{\tau=0}^{K-1} [F_{t+\tau}(y_i) - \epsilon_i] + 2B \sum_{\tau=0}^{K-1} \sum_{j=0}^{\tau-1} 1 \leq$$
$$Q_i(t) \sum_{\tau=0}^{K-1} [F_{t+\tau}(y_i) - \epsilon_i] + BK(K-1) \leq BK(K-1).$$

For (a) we take the upper bound for queue on the positive terms (Eq.(12)) and the lower bound on the queue for the negative terms (Eq.(11)), this gives an upper bound on the total. In (b) we rewrite the equation by bringing in the front the common $Q(t)$ terms. Next, at (c) we upper bound the non-$Q(t)$ terms with their norm and we pass the norm to every element. Finally, (d) follows by taking the upper bound $||F_t(y_i)||_1 \leq \sqrt{2B}$. Summing for I queues proves the lemma. ☐

We take Eq.10, where we drop (a) and replace (b) by Lem.(3). We sum the inequality for $t = \{1, 2, \dots, T-K\}$:

$$\underbrace{\sum_{\tau=0}^{K-1} \sum_{t=1}^{T-K} \Delta(t+\tau)}_{(c)} + \underbrace{V \sum_{\tau=0}^{K-1} \sum_{t=1}^{T-K} \left[C(\boldsymbol{x}^{t+\tau})) - C(\boldsymbol{y}) \right]}_{(d)} \leq$$
$$BK^2T + \underbrace{\alpha \sum_{\tau=0}^{K-1} \sum_{t=1}^{T-K} ||\boldsymbol{y} - \boldsymbol{x}^{t-1}||^2 - \alpha \sum_{\tau=0}^{K-1} \sum_{t=1}^{T-K} ||\boldsymbol{y} - \boldsymbol{x}^{t}||^2}_{(e)}.$$

To continue with the proof we need to lower bound the negative terms of the drift expression (c):

$$\sum_{\tau=0}^{K-1}\sum_{t=1}^{T-K}\Delta(t+\tau)=\frac{1}{2}\sum_{\tau=0}^{K-1}Q(T-K+\tau+1)^2-$$
$$\frac{1}{2}\sum_{\tau=0}^{K-1}Q(\tau+1)^2\geq-\frac{1}{2}\sum_{\tau=0}^{K-1}Q(\tau+1)^2\geq$$
$$-\frac{1}{2}\sum_{\tau=0}^{K-1}((\tau+1)\sqrt{2B})^2\geq-B\frac{K(K+1)(2K+1)}{6}.$$

The term (e) is bounded by $(e)\leq\alpha ID^2K$, this is easy to show by the cancellation of the terms of the telescopic sum and by dropping the negative terms, it is omitted due to space limitation. Finally, to complete the sum of regret K times we need to add and subtract terms in (d):

$$\sum_{\tau=0}^{K-1}\sum_{t=1}^{T-K}\left[C(\boldsymbol{x}^{t+\tau}))-C(\boldsymbol{y})\right]=K\sum_{t=1}^{T}\left[C(\boldsymbol{x}^{t}))-C(\boldsymbol{y})\right]-$$
$$\sum_{\tau=0}^{K-1}\sum_{t=1}^{\tau}\left[C(\boldsymbol{x}^{t}))-C(\boldsymbol{y})\right]-\sum_{\tau=0}^{K-1}\sum_{t=T-K+\tau+1}^{T}\left[C(\boldsymbol{x}^{t}))-C(\boldsymbol{y})\right]\geq$$
$$K\sum_{t=1}^{T}\left[C(\boldsymbol{x}^{t}))-C(\boldsymbol{y})\right]-DIc_iK(K-1).$$

where D is the max reservation, hence IKc_iD is the cost of assigning the maximum reservation for K iterations at all the I resources. By combining the results for (c),(d) and (e) and dividing by V and K we finish the proof of the lemma. □

In the next subsection, we give a corollary that summarizes the performance guarantees of THOR, utilizing the results of Th.(1) and Th.(2).

D. THOR has No Regret and is Feasible

The following corollary, combines the theoretical results of the previous subsections to prove the performance guarantees of THOR.

Corollary 1 (No Regret against $K=o(T)$)**.** *Fix $\varepsilon>0$, setting $\alpha=\frac{T^{\frac{3}{2}}}{V^{\frac{1}{2}}}$ and taking $K=T^{1-\varepsilon}$ and $V=T^{1-\frac{\varepsilon}{2}}$, THOR has no regret and is feasible:*

$$R_K(T)=O\left(T^{1-\frac{\varepsilon}{2}}\right),$$
$$Ctr(T)=O\left(T^{1-\frac{\varepsilon}{4}}\right).$$

Proof. We substitute in the expressions of Th.(1) and Th.(2) the values of α,K and V. Dropping the dominated terms completes the proof. □

We have proven that using THOR reservations we achieve ”no regret” against any K benchmark policy that has $K=o(T)$. This is an important finding that bridges the gap between [13], which proves ”no regret” against $K=1$ benchmark and [12], which proves that ”no regret” is impossible against T benchmark, with adversarial time varying constraints.

V. Numerical Evaluation

A. Google Cluster Data Analysis

In this subsection we compare the performance of our algorithm against an implementation of Follow The Leader (FTL), decribed in [7] and against the oracle best T slot reservation. The comparison is run on a public dataset from Google [14], [17]. The dataset contains detailed information about (i) measurements of actual usage of resources (ii) request for resources (iii) constraints of placing the resources in a big cluster comprised by 12500 machines. We will use the measurement of the actual usage of resources aggregated over the cluster. The time granularity of the measurements is 5 minutes for the period of 29 days and it is visualized in Fig. 1.

In Fig. 3 colored light gray is the time evolution of the workload demand sample path of the aggregate CPU (in subfig. 3a) and memory resources (in subfig. 3b). The blue line is THOR reservations, the red is the static oracle T-slot reservation and with dotted green the FTL reservations. In this experiment, ϵ_i is selected to be 10%. We see that for both resources, THOR predicts the resource reservation or quickly reacts to changes. On the other hand FTL is late to adapt, especially in the CPU case, causing many violations. In Table III, the above are expressed in numbers, THOR achieves significantly reduced violations while also achieving the lowest cost ($\sim 10\%$) less than the best static T-slot policy.

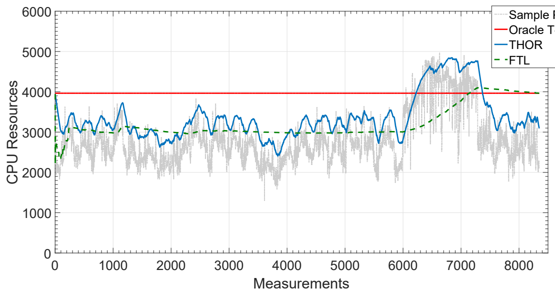
We run the same experiment, for a range of ϵ_i from very loose to very conservative violation guarantees. The numerical results are found in Tables I and II. The result shows that THOR always achieves the violation guarantee, but also that as the constraint gets stricter, THOR is becoming more protective than necessary. This is most probably due to overestimation of the $\boldsymbol{\Lambda}$ distribution that caps the adversary decisions in

TABLE I
CPU PERFORMANCE COMPARISON TABLE ACCORDING TO GUARANTEE

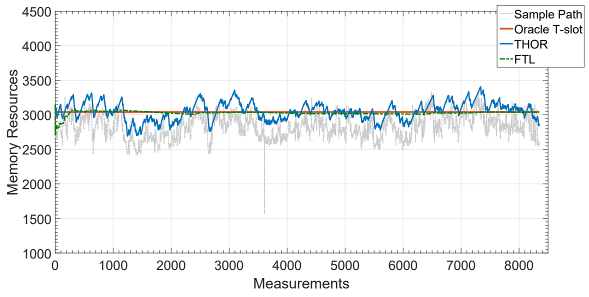
Guarantee	25%	20%	5%	1%	0.5%
T-Slot Vio	25.00	20.00	5.00	1.00	0.05
FTL Vio	36.28	31.29	13.63	5.41	3.57
THOR Vio	21.37	15.38	0.5	0	0
T-Slot Cost	2984	3124	4312	4682	4752
FTL Cost	2816	2896	3418	3740	3799
THOR Cost	3013	3193	3786	4238	4412

TABLE II
MEMORY PERFORMANCE COMPARISON TABLE ACCORDING TO GUARANTEE

Guarantee	25%	20%	5%	1%	0.5%
T-Slot Vio	25.00	20.00	5.00	1.00	0.05
FTL Vio	26.38	21.43	6.94	2.61	1.64
THOR Vio	16.38	10.95	0.4	0	0
T-Slot Cost	2943	2970	3096	3189	3225
FTL Cost	2937	2962	3071	3129	3159
THOR Cost	2968	3006	3181	3258	3293



(a) CPU Reservations



(b) Memory Reservations

Fig. 3. Comparison Plots of reservation updates for different policies. Light gray on the background is the sample path of the actual resources required at the server. From the subfigures we can see that in the non-stationary case of CPU resource requirement, FTL policy is lagging significantly in resource reservation, incurring many more violations than the maximum 10%. For memory we can see that FTL is similar to the best static, while our algorithm achieves better performance by tracking the fluctuations. See Table III for the numerical comparison.

TABLE III
POLICY COMPARISON TABLE, $\epsilon = 10\%$

Performance	T-slot Oracle	FTL	THOR
Average CPU Cost	3964	3203	3365
Average Violations (%)	10.00	21.41	5.64
Average MEM cost	3041	3054	3027
Average Violations (%)	10.00	11.84	3.7

theory, which is estimated by random sampling of max values. Furthermore, FTL again struggles to achieve the constraint in the whole range of guarantees, with slightly better performance in memory reservation. Even though THOR is more protective than necessary cost wise the performance is very similar to the best T-slot static policy, with the maximum difference to be less than 4% for very loose guarantee and actually achieving better performance on the tough CPU reservation, especially as the guarantee gets stronger.

VI. CONCLUSION

In this paper we introduce THOR, an online resource reservation policy for clouds. Cloud environments are very challenging due to volatile and unpredictable workloads. THOR uses (i) predictor queues and (ii) drift plus penalty plus smoothness, to fine tune reservations, such that overprovisioning is minimized while underprovisioning is maintained below a guaranteed ϵ_i parameter chosen by the system administrator. We prove that THOR is feasible in a growing horizon T and that it achieves similar performance to any sublinear to T, K-slot oracle static reservation. In simulation results using a public dataset from Google cluster, we vastly outperform the heavy to implement in practice follow the leader algorithm and perform similarly to the T-slot oracle policy.

REFERENCES

[1] M. J. Neely, "Stochastic Network Optimization with Application to Communication and Queueing Systems," *Synthesis Lectures on Communication Networks*, 2010.

[2] M. Zinkevich, "Online Convex Programming and Generalized Infinitesimal Gradient Ascent," ser. ICML, 2003.

[3] R. B. Q. Zhang, M. F. Zhani and J. L. Hellerstein, "Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud," *IEEE Transactions on Cloud Computing*, Jan 2014.

[4] "How AWS Pricing Works," Amazon, 2018. [Online]. Available: https://aws.amazon.com/whitepapers/

[5] N. C. Petruzzi and M. Dada, "Pricing and the Newsvendor Problem: A Review with Extensions," *Operations research*, 1999.

[6] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis," in *ACM Symposium on Cloud Computing (SoCC)*, San Jose, CA, USA, Oct. 2012.

[7] S. Shalev-Shwartz, "Online Learning and Online Convex Optimization," *Foundations and Trends® in Machine Learning*, 2012.

[8] R. N. Elena Veronica Belmega, Panayotis Mertikopoulos and L. Sanguinetti, "Online Convex Optimization and No-Regret Learning: Algorithms, Guarantees and Applications," 2018. [Online]. Available: http://arxiv.org/abs/1804.04529

[9] M. Mahdavi, R. Jin, and T. Yang, "Trading Regret for Efficiency: Online Convex Optimization with Long Term Constraints," *Journal of Machine Learning Research*, 2012.

[10] R. Jenatton, J. Huang, and C. Archambeau, "Adaptive Algorithms for Online Convex Optimization with Long-Term Constraints," *arXiv preprint arXiv:1512.07422*, 2015.

[11] J. D. Abernethy, E. Hazan, and A. Rakhlin, "Interior-Point Methods for Full-Information and Bandit Online Learning," *IEEE Transactions on Information Theory*, 2012.

[12] S. Mannor, J. N. Tsitsiklis, and J. Y. Yu, "Online Learning with Sample Path Constraints," *Journal of Machine Learning Research*, 2009.

[13] M. J. Neely and H. Yu, "Online Convex Optimization with Time-Varying Constraints," *arXiv preprint arXiv:1702.04783*, 2017.

[14] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google Cluster-Usage Traces: Format + Schema," Google Inc., Mountain View, CA, USA, Technical Report, Nov. 2011, revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[15] M. J. N. Leonidas Georgiadis and L. Tassiulas, "Resource Allocation and Cross-Layer Control in Wireless Networks," *Foundations and Trends® in Networking*, 2006.

[16] N. Parikh, S. Boyd *et al.*, "Proximal Algorithms," *Foundations and Trends® in Optimization*, 2014.

[17] J. Wilkes, "More Google Cluster Data," Google Research Blog, Nov. 2011.