Performance Evaluation 162 (2023) 102376

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

POBO: Safe and optimal resource management for cloud microservices

Hengquan Guo[1], Hongchen Cao[1], Jingzhu He *, Xin Liu *, Yuanming Shi

ShanghaiTech University, China

ARTICLE INFO

Keywords:
Microservice resource management
Service-Level Agreement
Safe Bayesian optimization
Primal–dual optimization
Penalty-based design

ABSTRACT

Resource management in microservices is challenging due to the uncertain latency–resource relationship, dynamic environment, and strict Service-Level Agreement (SLA) guarantees. This paper presents a Pessimistic and Optimistic Bayesian Optimization framework, named POBO, for safe and optimal resource configuration for microservice applications. POBO leverages Bayesian learning to estimate the uncertain latency–resource functions and combines primal–dual and penalty-based optimization to maximize resource efficiency while guaranteeing strict SLAs. We prove that POBO can achieve sublinear regret and SLA violation against the optimal resource configuration in hindsight. We have implemented a prototype of POBO and conducted extensive experiments on a real-world microservice application. Our results show that POBO can find the safe and optimal configuration efficiently, outperforming Kubernetes' built-in auto-scaling module and the state-of-the-art algorithms.

1. Introduction

1.1. Motivation

The rapid emergence of microservices has revolutionized how software applications are designed and managed. Microservices are loosely coupled, enabling better modularity, improved fault isolation, easier scaling, and more agile development in technology choices compared to traditional monolithic architectures [1]. Leading cloud service providers, such as Google and Amazon, often offer off-the-shelf microservice architecture for users to build their applications [2–4].

Despite the benefits of microservice architecture, resource management for microservice applications faces several challenges. First, the requests' latencies are affected by the amount of allocated resources; nevertheless, such a relationship is uncertain. For example, Fig. 1(a) plots the P90 latency of three different requests versus the number of allocated containers. We observe that different types of requests show different latency–resource relationships. For each type of request, when more containers are used, the P90 latencies decrease nonlinearly. When reaching the convergent point (e.g., 101.81 ms of P90 latency when 20 containers are used for the login request), the P90 latencies do not decrease anymore, even though we add more resources to the microservice. Accurately understanding and learning the latency–resource relationship and finding an optimal configuration remains a challenge. Besides that, the heterogeneity of requests' arrival patterns introduces another level of uncertainty to learning the function.

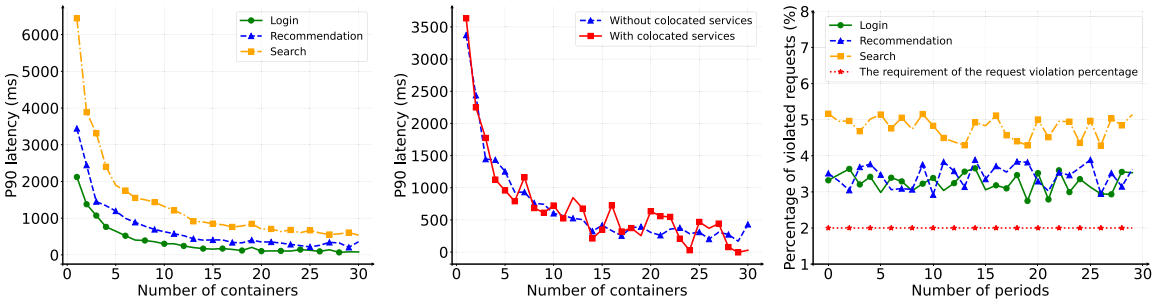
Second, microservices usually run in a dynamic environment because multiple services are co-located on a common cloud server in a real-world system. Fig. 1(b) illustrates the latency–resource function for one microservice with and without co-located running

* Corresponding authors.
E-mail addresses: guohq@shanghaitech.edu.cn (H. Guo), caohch1@shanghaitech.edu.cn (H. Cao), hejzh1@shanghaitech.edu.cn (J. He), liuxin7@shanghaitech.edu.cn (X. Liu), shiym@shanghaitech.edu.cn (Y. Shi).

[1] These authors contributed equally to this work.

https://doi.org/10.1016/j.peva.2023.102376

Available online 10 October 2023



(a) Uncertain latency-resource relationship. (b) Dynamic environment. (c) Strict SLA.

Fig. 1. The three challenges of achieving safe and efficient configuration in microservices: uncertain latency–resource relationship, dynamic environment, and strict SLA requirement. (a) The correlation between the number of containers and P90 latencies across various requests. (b) The P90 latency curves with and without co-located services. (c) The percentage of violated requests by using the Kubernetes auto-scaler.

services. We observe that the function fluctuates more in the presence of other co-located services compared with the one without any co-located services. In other words, microservices' dynamic runtime environment poses a challenge in learning the latency–resource function, making it even more difficult to find the optimal resource configuration efficiently.

Third, the availability of microservices is determined by a user-defined Service-Level Agreement (SLA), which is usually strict since users always expect short response times for requests and a small percentage of requests violating a user-defined response time requirement. For example, Fig. 1(c) shows that by using Kubernetes' built-in auto-scaler for three types of requests, there are 3% or more requests have response times that exceed the user-defined requirement (i.e., 300 ms) on average, while users expect that the percentage is less than 2%. In addition, it is difficult for users, even experienced developers, to come up with the optimal configuration of the auto-scaler while satisfying the SLA requirements. The dilemma is that (1) by using fewer resources for microservices, the SLA requirements are easier to violate, and (2) by using more resources for microservices, there is a higher chance of causing resource waste while the requests' response times cannot be shortened further. It is challenging to understand such tradeoffs and to find the optimal solution while ensuring SLA guarantees strictly.

To address these challenges, we aim to develop a general framework that can accurately learn the latency–resource functions in the dynamic environment and quickly identify the most efficient configuration without prior knowledge of requests' arrival patterns while ensuring SLA guarantees.

In this paper, we study the problem of microservice resource management, where we optimize resource utilization while considering SLA constraints on average system-wide request latency and tail latency for latency-sensitive applications, respectively. The SLA on the system-wide request's latency targets a decent service across all requests. The SLA on the tail latency is for the latency-sensitive applications that have priority. To tackle this problem, we introduce a Bayesian optimization-based approach to learn the inherent complexity and uncertainty in the microservice architecture. Then, we propose a primal–dual and penalty-based optimization to explore various resource configurations safely. Our approach acts in an online manner without offline performance profiling and can quickly converge to a safe and efficient configuration. Besides, we want to emphasize that our approach is very flexible to incorporate other types of constraints beyond the current constraints on the system-wide latency or tail latency.

1.2. Related work

Microservices resource management. Resource management in microservices has gained significant attention in the literature. GRAF [5] employed graph neural networks for predicting latency and optimized resource efficiency and SLA satisfaction using a handcrafted designed penalty coefficient. Sinan [6] also trained ML-based models to predict the latency performance and designed a rule-based scheduler to meet SLA requirements. Erms [7] approximated the latency of a microservice application with a piece-wise linear function w.r.t. the workload, resource usage, and interference and used it to design resource-efficient scheduling policies with SLA guarantees. These works fall into a category of "off-profile then optimize" approaches, and SLA requirements might fail to be satisfied in the dynamic environment.

(Safe) Bayesian optimization for resource management. Bayesian optimization (BO) has been widely used for microservice management in searching for the best resource configuration [8–11]. RAMBO [8] used a BO approach to construct an acquisition function that combines multiple objectives and optimizes multiple metrics simultaneously (e.g., throughput, resource consumption). CLITE [9] also used BO with a manually crafted objective function to achieve QoS (Quality of Service) for latency-critical requests. These studies show the potential of BO in efficient microservice resource management. However, they cannot model the noisy and dynamic nature in microservice infrastructures [12]. Aquatope [12] used an advanced Bayesian neural network to account for the dynamic interference and aimed at satisfying SLA on the average latency instead of SLA on tail latency.

Safe/Constrained BO is a proper framework for the microservice resource configuration while guaranteeing SLA, where SLA requirements are naturally formulated as the constraints [11–13]. A common approach in these works is to construct a safe/feasible set within which an optimal decision can be searched for. However, the approach suffers from high computational complexity and

has to assume a known safe decision in advance. Recently, two lightweight algorithms have been proposed to solve safe BO, CKB in [14] and RPOL in [15]. The work [14] established the sublinear regret and average constraint violation, and [15] established the sublinear regret and cumulative constraint violation. However, neither work can guarantee average and cumulative violation simultaneously. Besides, the proposed algorithms only target optimizing the performance metrics for the systems serving single-type requests.

In this paper, we propose POBO, a lightweight algorithm for safe and optimal resource configuration for microservice management. Compared to these existing works, our algorithm can guarantee SLA requirements on both system-wide latency and tail latency. Theoretically speaking, our work establishes a sublinear regret while minimizing both average constraint violation (for system-wide latency) and cumulative violation (for tail latency) simultaneously. Moreover, our algorithm is applicable to systems serving multiple-type requests coexistence and does not require any prior knowledge of the distribution of arriving requests.

1.3. Main contribution

In this paper, we study the problem of microservice resource management with SLA guarantees. Our objective is to design a sequence of resource configurations $\{x_1, x_2, \dots, x_T\}$ to maximize resource utility $\mathbb{E}[\sum_{t=1}^{T} f(c_t, x_t)]$ over T periods while satisfying the system-wide latency constraint $\mathbb{E}[\sum_{t=1}^{T} g(c_t, x_t)] \leq 0$ and the tail latency constraints $G(c_t, x_t) \leq 0$ for the request c_t in the priority set $\mathcal{C}_p$. As the functions $f(\cdot,\cdot), g(\cdot,\cdot)$, and $G(\cdot,\cdot)$ are black-box functions and potentially very complicated, it is unavoidable to have SLA violations: the system-wide latency violation (or average violation), $\mathbb{E}[\sum_{t=1}^{T} g(c_t, x_t)]$ and the tail latency violation (or cumulative violation), $\mathbb{E}[\sum_{t=1}^{T} G^+(c_t, x_t)\mathbb{I}\{c_t \in \mathcal{C}_p\}]$. We introduce POBO, an efficient resource configuration framework to maximize resource efficiency while keeping the SLA violation minimal. Our contribution can be summarized as follows:

- **Algorithm Design:** POBO adopts optimistic and pessimistic Bayesian learning for black-box functions and leverages primal–dual and penalty-based optimization to synthesize a single surrogate function to simultaneously control the system-wide latency and tail latency. This design can automatically balance resource efficiency and SLA violation.
- **Theoretical Analysis:** We establish that POBO achieves $O(\gamma_T\sqrt{T})$ regret, $O(1)$ average violation, and $O(\gamma_T\sqrt{T})$ cumulative violation over T periods, which matches the best average violation in [14] and the best cumulative violation in RPOL [15]. To prove the theoretical results, we establish a novel Lyapunov drift analysis that incorporates the penalized constraint functions, which enables a unified analysis of "regret + average constraint + cumulative violation" as a whole.
- **Implementation:** We have implemented a prototype of POBO and conducted extensive experiments on a real-world microservice application. Our experimental results show that POBO significantly outperforms the Kubernetes built-in HPA auto-scaler and the state-of-the-art algorithm CKB. POBO can effectively adjust resource configurations in response to incoming requests. The results also demonstrate that POBO can swiftly converge to the optimal policy without prior knowledge of the distribution of multiple-type requests, thereby outperforming other scaling mechanisms. Lastly, the results show POBO's robustness in dynamic environments, proving that POBO is applicable in a broader setting.
- **Code Availability:** The source code of POBO is available at https://github.com/caohch-1/POBO.

The rest of the paper is organized as follows. Section 2 introduces the problem formulation of microservice resource management. Section 3 discusses the design of POBO. Section 4 presents the theoretical results and analysis. Section 5 presents the experimental evaluation. Section 6 concludes the paper.

2. Microservice resource management: a constrained Bayesian optimization approach

In this section, we introduce the problem of microservice resource management and formulate it via constrained Bayesian optimization. We study a microservice system with a pool of containers where a stream of requests arrives continuously. At period t, a c_t-type request ($c_t \in \mathcal{C}$) arrives according to an *unknown* underlying distribution $\mathbb{P}(\cdot)$. The resource manager decides a resource configuration x_t for the request. When the request c_t is completed at period t, we observe the (noisy) feedback information for the configuration x_t, such as the residual resources, average latency, and tail latency. In the microservice system, the resource manager must maintain a low system-wide latency while guaranteeing a strict SLA for latency-critical requests. Given the configuration x_t for the c_t-type microservice request, we model $f(c_t, x_t)$ to be the resource utility (e.g., residual resources); $g(c_t, x_t)$ to be the average performance metric (e.g., average latency); and $G(c_t, x_t)$ to be the critical performance metric (e.g., P90 tail latency). Due to the dynamic nature of the microservice system, we only have access to the noisy feedback as follows

$$\begin{aligned} r_t &= f(c_t, x_t) + \eta_t, \\ v_t &= g(c_t, x_t) + \xi_t, \\ w_t &= G(c_t, x_t) + \tau_t, \ \forall c_t \in \mathcal{C}_p, \end{aligned}$$

where η_t, ξ_t, and τ_t denote the noise. The problem of microservice resource management can be formulated in the following

$$\max_{x_t \in \mathcal{X}} \ \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} f(c_t, x_t)\right] \tag{1}$$

$$\text{s.t.} \ \ \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} g(c_t, x_t)\right] \leq 0, \tag{2}$$

$$G(c_t, x_t) \le 0, \ \forall c_t \in C_p, \tag{3}$$

where the objective (1) denotes the average resource utility; (2) denotes the average system-wide latency constraint; and (3) denotes the tail latency constraint for latency-sensitive (priority) requests.

Given the full knowledge of the system (i.e., the statistics of arrival requests $\mathbb{P}(\cdot)$, resource utility function $f(\cdot,\cdot)$, and latency functions $g(\cdot,\cdot)$ and $G(\cdot,\cdot)$, one could compute an offline optimal resource configuration by solving (1)–(3). However, as discussed in the section of Introduction, it is extremely challenging (if not impossible) to obtain this accurate information because of the uncertain, dynamic, and noisy nature of the microservice systems. Therefore, we propose a learning-based approach to address these challenges.

2.1. Bayesian optimization

Since $f(\cdot,\cdot)$, $g(\cdot,\cdot)$, and $G(\cdot,\cdot)$ are unknown apriori and complicated (latency functions g and G have very complicated relationship w.r.t. the request structure and configuration [7,10,16]), we model them with Gaussian processes (GPs) via a Bayesian perspective to tackle the uncertainty and noise. Technically, we assume that the reward and constraint functions lie within a Reproducing Kernel Hilbert Space (RKHS) with a bounded norm such that these black-box functions can be modeled properly with GPs. This regular assumption is reasonable in the context of microservice resource management, as it captures the inherent continuity of the underlying functions as shown in Fig. 1.

Gaussian process model for reward and constraints functions: Gaussian process (GP) is a random process including a collection of random variables that follows a joint Gaussian distribution. A Gaussian process $\mathcal{GP}(\mu(x), k_t(x,x'))$ over $\mathcal{X}$ is specified by its mean $\mu(x)$ and covariance $k(x,x')$. For a type-c request, we introduce its GP for the reward function $f(c,x)$ and GPs for g and G are similar.

We define $\mathcal{GP}(\mu^f(c,x), k^f(c,x,x'))$ for $f(c,x)$ such that $\mu^f(c,x) = \mathbb{E}[f(c,x)]$ and $k^f(c,x,x') = \mathbb{E}[(f(c,x) - \mu^f(c,x))(f(c,x') - \mu^f(c,x'))]$. Let $\mathcal{A}_t = \{x_1, \dots, x_{t-1}\}$ be the collection of decisions and $\{r_1, \dots, r_{t-1}\}$ be the collection of noisy reward feedback until tth period, respectively. The posterior distribution $\mathcal{GP}(\mu_t^f(\cdot,\cdot), k_t^f(\cdot,\cdot,\cdot))$ updates at the beginning of period t

$$\mu_t^f(c,x) = \ k_t^f(c,x)^T (V_t^f(c,\lambda))^{-1} r_{1:t} \tag{4}$$

$$k_t^f(c,x,x') = \ k^f(c,x,x') - k_t^f(c,x)^T (V_t^f(c,\lambda))^{-1} k_t^f(x'), \tag{5}$$

$$\sigma_t^f(c,x) = \ \sqrt{k_t^f(c,x,x)}, \tag{6}$$

where $K_t^f(c) := [k^f(c,x,x')]_{x,x' \in \{x_1,\dots,x_{t-1}\}}$, $V_t^f(c,\lambda) := K_t^f(c) + \lambda I$, $\lambda = 1 + 2/T$, $r_{1:t} = [r_1, \dots, r_{t-1}]$, and $k_t^f(c,x) := [k^f(c,x_1,x), \dots, k^f(c,x_{t-1},x)]^T$. Similarly, we define GP models for the constraint function g and G to be $\mathcal{GP}(\mu_t^g(c,x), k_t^g(c,x,x'))$ and $\mathcal{GP}(\mu_t^G(c,x), k_t^G(c,x,x'))$ with the mean $\mu_t^g(c,x)$ and $\mu_t^G(c,x)$ and the covariance $k_t^g(c,x,x')$ and $k_t^G(c,x,x')$, respectively. The models for g and G update the same as in (4)–(6). The kernel function is designed by choice, and one popular kernel is the square exponential (SE) kernel $k_{\text{SE}}(c,x,x') = e^{\frac{-\|x-x'\|^2}{2u^2}}, \forall c$, where $u > 0$ is a positive hyper-parameter.

With the estimated mean and variance in (4) and (6), we construct a confidence interval for the objective function $f(c,x)$. Specifically, we design a proper β_t^f such that $f(c,x)$ lies within its lower confidence bound (LCB) and upper confidence bound (UCB)

$$[\mu_t^f(c,x) - \beta_t^f \sigma_t^f(c,x), \mu_t^f(c,x) + \beta_t^f \sigma_t^f(c,x)]$$

with a high probability according to [17]. Moreover, we have the similar confidence intervals with proper β_t^g and β_t^G such that $[\mu_t^g(c,x) - \beta_t^g \sigma_t^g(c,x), \mu_t^g(c,x) + \beta_t^g \sigma_t^g(c,x)]$ for g function and $[\mu_t^G(c,x) - \beta_t^G \sigma_t^G(c,x), \mu_t^G(c,x) + \beta_t^G \sigma_t^G(c,x)]$ for G function. For the simple exposition, let the parameters of the Gaussian processes be $\Theta_t^f = (\mu_t^f, \sigma_t^f, k_t^f)$, $\Theta_t^g = (\mu_t^g, \sigma_t^g, k_t^g)$, and $\Theta_t^G = (\mu_t^G, \sigma_t^G, k_t^G)$.

For unconstrained Bayesian optimization, one could utilize the principle of optimism in the face of uncertainty (OFU) and design the upper confidence bound (UCB) algorithm to address the exploration–exploitation dilemma. Specifically, the GP-UCB algorithm decides a configuration such that

$$x_t = \arg\max_{x \in \mathcal{X}} \mu_t^f(c,x) + \beta_t^f \sigma_t^f(c,x).$$

However, under (black-box) SLA constraints, we need to safely explore the configuration space with as small SLA violation as possible. This requires a careful balance between resource efficiency and SLA. Next, we introduce POBO to achieve this goal.

3. System and algorithm design

In this section, we present the design of the POBO framework. We first illustrate the system design and then describe the resource management algorithm.

3.1. Framework overview

Fig. 2 gives an overview of how POBO works. When receiving a request from the user, POBO sends the request to the ***Learning*** module to estimate the request latency w.r.t. a variety of resource configurations. The ***Decision*** module determines the best resource configuration for the request and adjusts the number of containers via the resource manager. After the request is completed, POBO collects the feedback, including Service-Level Agreement (SLA) violation and tail latency, which is used to update ***Learning*** and ***Update*** modules for the next period.

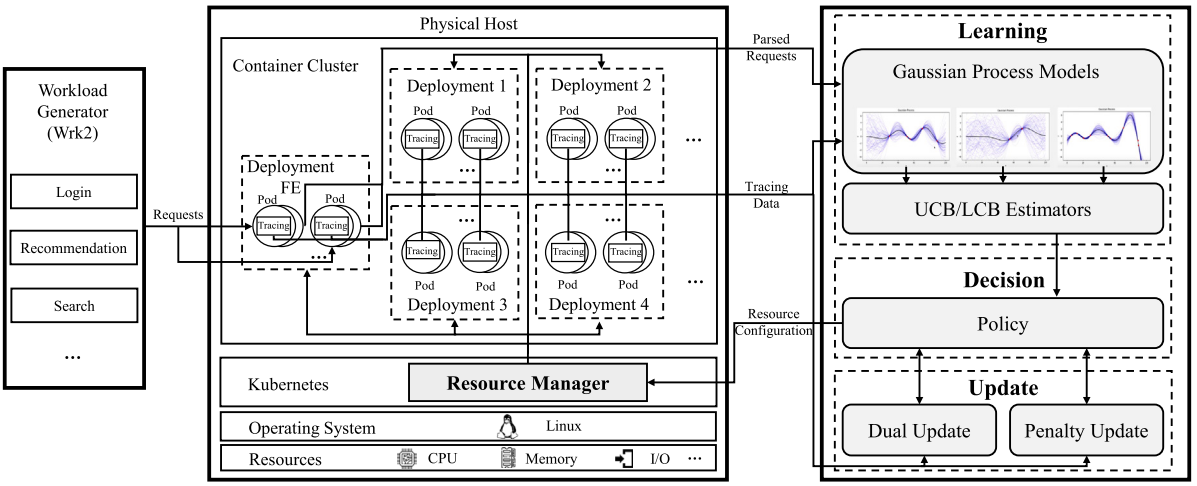


Fig. 2. The system architecture of POBO.

```
{   'traceID': '25d872470d79fd55',
    'spans': [...{...
                    'operationName': 'HTTP GET /recommendations',
                    ...
                    'startTime': 1684827732697635,
                    'duration': 1101330,
                    'tags': [...
                            {   'key': 'http.url',
                                'type': 'string',
                                'value': '/recommendations?require=dis\u0026lat=38.209\
                                    u0026lon=-121.941'},
                            ...],...},
                {   'traceID': '25d872470d79fd55',
                    'spanID': '0e5acf845ed3e1fc',
                    ...},...],
    ...}
```

Fig. 3. A request's trace example.

3.2. System design

As shown in Fig. 2, the container orchestration platform, i.e., Kubernetes, manages a cluster of pre-warmed containers. Since microservice applications are highly decoupled, each unit service is typically deployed and managed as a separate deployment. For example, the login service is provided by one deployment, and the recommendation service is provided by another deployment. For each deployment, pods are the minimum units that Kubernetes can create and manage. A pod is defined as a group of one or more containers with shared storage and network resources and a specification on how to run the containers. Initially, all deployments of our microservice application use one pod by default. Each deployment has a fixed maximum number of pods to deliver the unit service. Within the deployment, each container is pre-warmed, which means that each container is already installed with the same image that delivers the unit service before the microservice application runs. The advantage is that by simply activating or deactivating containers of several pods, POBO changes the amount of allocated resources to quickly react to the workload change. In comparison, for cold start approaches, it is essential to create or destroy a pod and its container(s) from scratch to achieve scaling, which causes considerable delays before requests are processed, further prolonging response times.

The user, i.e., the workload generator, sends the request to the front end of microservice applications continuously. Note that the front-end unit service is achieved by the pods within the front-end deployment, i.e., Deployment FE in Fig. 2. Once an incoming request is received, the pods parse the request and send useful information, including the request type c_t, to the ***Learning*** module. The primary goal of the learning model is to give the estimated request latency based on the request type. For each request type, POBO maintains a particular Gaussian process model and estimates the latency via UCB/LCB learning. After that, the estimated request latency is fed into the ***Decision*** module to determine the optimal resource configuration. We implement a resource manager module that is plugged into Kubernetes. Based on the received resource configuration, the resource manager adjusts the number of running pods to process the request within the corresponding deployment.

As shown in Fig. 2, we deploy a tracing module within each container to record information about the request. Fig. 3 presents a request's tracing example well-structured in JSON format. Each trace has its own unique trace ID. The trace is formed into the hierarchical tree structure. For example, each trace consists of multiple spans where the `'traceID'` is viewed as the root and the

'spanID' is viewed as the leaf. We can drill down the tree path to find the deployment where the trace example is collected. The trace consists of multiple fields. The 'operationName' and 'tag' fields show the request type and the associated URL link. The 'startTime' field indicates the timestamp when the request arrives at the pod, and the 'duration' field indicates the execution time to process the request, also viewed as the latency. Based on the collected latencies during one period, we can easily get the Service-Level Agreement (SLA) violation based on user-defined response time requirement and tail latency based on the requests' latency distribution.

We use Jaeger [18] to implement end-to-end distributed tracing because (1) Jaeger is a generic tracing tool that is agnostic to microservice applications and programming languages; (2) the collected request latencies are accurate; and 3) the runtime overhead is negligible for each tracing module. However, the microservice applications are usually built on top of tens or hundreds of containers and each of which contains a tracing module. To reduce the overhead imposed by multiple tracing modules, we adopt the down-sampling strategy to trace the requests selectively. For example, POBO only traces 50 requests when 100 requests are received and processed. We tune the sampling rate to reduce the overall overhead to less than 3.5%, which makes POBO practical for use in production systems.

After the request is processed, POBO sends the traced request latency to the ***Learning*** module to update the GPs model between the resource configuration and latency for each request type. Moreover, when one period elapses, POBO sends the SLA violation and tail latency to the ***Update*** module consisting of *Dual update* and *Penalty update* to update the cumulative SLAs violation.

3.3. Online resource management algorithm

In this section, we propose a pessimistic and optimistic Bayesian optimization (POBO) framework to address the problem of microservice resource management. POBO learns the reward function $\hat{f}_t$ optimistically, and the constraint functions $\check{g}_t$ and $\check{G}_t$ pessimistically. Motivated by the primal–dual and penalty-based optimization approach, POBO synthesizes a surrogate function to find a safe and efficient configuration to maximize the reward (resource utilization) while guaranteeing minimal constraint violations (SLAs).

Learning: When a c_t−type microservice request arrives, we construct optimistic/pessimistic estimation for the reward/constraints functions via UCB/LCB learning with the GP parameters Θ_t^f, Θ_t^g, and Θ_t^G

$$\hat{f}(c_t,x)=\left[\mu_t^f(c_t,x)+\beta_t^f\sigma_t^f(c_t,x)\right]_{-B_f}^{B_f}, \tag{7}$$

$$\check{g}(c_t,x)=\left[\mu_t^g(c_t,x)-\beta_t^g\sigma_t^g(c_t,x)\right]_{-B_g}^{B_g}, \tag{8}$$

$$\check{G}(c_t,x)=\left[\mu_t^G(c_t,x)-\beta_t^G\sigma_t^G(c_t,x)\right]_{-B_G}^{B_G}. \tag{9}$$

The $[z]_l^h$ is the projection operator such that z is within the interval of $[l,h]$. The optimistic–pessimistic design is to encourage exploration and to avoid over-pessimistic decisions during the initial period.

Decision: Based on the learned $\hat{f}(c_t,x)$, $\check{g}(c_t,x)$, and $\check{G}(c_t,x)$, POBO utilizes primal–dual and penality-based optimization in designing a single surrogate function in Eq. (10). Specifically in Eq. (10), we have the dual variable Q_t with $\check{g}(c,x)$ and the penalty factor $\hat{Q}_t$ with $\check{G}(c,x)$ to control the system-wide latency violation and tail latency violation, respectively. Moreover, the adaptive learning rates V_t in POBO can establish a good trade-off between resource efficiency and SLA violation.

Update: After the configuration x_t is deployed for the c_t- request, we observe the noisy feedback of reward $r_t(c_t,x_t)$ and SLA violation $v_t(c_t,x_t)$ and $w_t(c_t,x_t)$, if c_t is a priority request (i.e., $c_t\in C_p$). These observations are used to update GP models according to (4)–(6) (called a learning strategy $\mathcal{U}$) and the dual variable Q_t in Eq. (11) and the penalty variable $\hat{Q}_t$ in Eq. (12). Intuitively, the dual and penalty variables increase when the resource x_t is not sufficient to support the SLA, which in turn serves as indicators for future configuration.

We summarize POBO in the following and then provide the underlying intuition.

POBO Framework

Initialization: $Q_1=0$, $\hat{Q}_1=1$, $V_t=\frac{\delta\sqrt{t}}{8B_f}$, $\epsilon_t=\frac{6\beta_T^g\sqrt{\gamma_T^g+2}}{\sqrt{t}}$, $\eta_t=t$, Θ_1^f, Θ_1^g and Θ_1^G.
For $t=1,\cdots,T$,

- **Pessimistic–optimistic learning:** observe a c_t-type microservice request, estimate the reward function $\hat{f}_t(c,x)$ and the cost function $\check{g}_t(c,x)$, $\check{G}_t(c,x)$ according to GP-UCB/LCB with $(\Theta_t^f,\Theta_t^g,\Theta_t^G)$ in (7)-(9).
- **Rectified penalty-based decision:** select a configuration x_t such that

$$\underset{x\in\mathcal{X}}{\operatorname{argmax}}\ \hat{f}_t(c_t,x)-\frac{Q_t\check{g}_t(c_t,x)}{V_t}-\hat{Q}_t\check{G}_t^+(c_t,x)\mathbb{I}\{c_t\in C_p\} \tag{10}$$

- **Feedback:** noisy reward $r_t(c_t,x_t)$, SLA violations $v_t(c_t,x_t)$ and $w_t(c_t,x_t)$ if c_t is a priority request (i.e. $c_t\in C_p$).
- **Dual update:**

$$Q_{t+1}=\left(Q_t+\check{g}_t(c_t,x_t)+\epsilon_t\right)^+. \tag{11}$$

- **Penalty update:**

$$\hat{Q}_{t+1} = \max\left(\hat{Q}_t + (w_t(c_t, x_t))^+ \mathbb{I}\{c_t \in C_p\}, \eta_t\right). \tag{12}$$

- **Model update:**

$$\Theta^f_{t+1} = \mathcal{U}(\Theta^f_t, \{c_t, x_t, r_t\}),\ \Theta^g_{t+1} = \mathcal{U}(\Theta^g_t, \{c_t, x_t, v_t\}),\ \Theta^G_{t+1} = \mathcal{U}(\Theta^G_t, \{c_t, x_t, w_t\}).$$

We provide the intuition behind POBO. Recall the offline problem in (1)–(3), we decompose it into the problem for an individual request (c_t-type request) at every period

$$\begin{aligned} \max_{x\in\mathcal{X}}\ & f(c_t, x) \\ \text{s.t.}\ & g(c_t, x) \le 0, \\ & G(c_t, x) \le 0,\ \forall c_t \in C_p. \end{aligned}$$

The problem above can be solved by optimizing the Lagrange function

$$L(c_t, x, \lambda, \vartheta) := f(c_t, x) - \lambda g(c_t, x) - \vartheta G(c_t, x),$$

where λ and ϑ are the dual variables corresponding to the constraint functions g and G, respectively. As the constraint function G has priority and is supposed to be satisfied for every request, we impose a rectified operator on G^+ to induce a feasible decision for the G function. The dual variables λ and ϑ are approximated with Q_t/V_t (a scaled version of Q_t) and $\hat{Q}_t$, respectively. Since all functions are unknown and approximated via Gaussian processes, we use an optimistic version $\hat{f}_t(c, x)$ of $f(c, x)$ and pessimistic version $\check{g}(c, x)$ and $\check{G}(c, x)$ of $g(c, x)$ and $G(c, x)$. Combine all these ingredients, we finally have the surrogate function

$$L(c_t, x, \lambda, \vartheta) := \hat{f}(c_t, x) - \frac{Q_t \check{g}(c_t, x)}{V_t} - \hat{Q}_t G^+(c_t, x).$$

The term Q_t is a proxy for the dual variable λ and captures the constraint violation g function in the long term. Its update is

$$Q_{t+1} = \left(Q_t + \check{g}_t(c_t, x_t) + \varepsilon_t\right)^+.$$

Intuitively, a positive value of Q_t indicates the system-wide latency violates until the period t. We add an extra term ε_t such that the decision becomes slightly conservative to satisfy the average system-wide constraints. The term $\hat{Q}_t$ is a strict proxy for the dual variable ϑ and captures the anytime violation G function for every control period, its update is

$$\hat{Q}_{t+1} = \max\left(\hat{Q}_t + (w_t(c_t, x_t))^+ \mathbb{I}\{c_t \in C_p\}, \eta_t\right).$$

The value of $\hat{Q}_t$ keeps increasing as long as the tail latency violates at a period. Note we also add an extra term η_t, which imposes at least the amount of η_t penalty for the period t and forces the decision to satisfy the tail latency constraint. Moreover, the learning rate V_t is designed to control the tradeoff between regret and constraint violation in Theorem 1.

4. Theoretical analysis

In this section, we present the theoretical analysis of POBO for microservice resource management. We first define the information gain at period t to be $\gamma^f_t := \max_{\mathcal{A}_t\in\mathcal{X}:|\mathcal{A}_t|=t-1} \frac{1}{2}\ln|I + \lambda^{-1}K^f_t|$, $\gamma^g_t := \max_{\mathcal{A}_t\in\mathcal{X}:|\mathcal{A}_t|=t-1} \frac{1}{2}\ln|I + \lambda^{-1}K^g_t|$ and $\gamma^G_t := \max_{\mathcal{A}_t\in\mathcal{X}:|\mathcal{A}_t|=t-1} \frac{1}{2}\ln|I + \lambda^{-1}K^G_t|$, which are important parameters in GP learning. They depend on the choice of the kernel function and the domain $\mathcal{X}$, and would play a key role in our following regret and violation analysis. For SE kernel function, we have γ^f_t, γ^g_t and γ^G_t are $O((\ln(t))^{d+1})$ if $\mathcal{X}$ is compact and convex with dimension d.

Next, we establish the regret, average violation for the system-wide latency constraint, and cumulative violation for the tail latency constraint.

Regret and constraint violation: Recall the offline optimization problem (1)–(3) in Section 2, given the complete knowledge of (1)–(3), we can compute its optimal solution x^*, which is used to define regret and constraint violation in the following

$$\mathcal{R}(T) := T f(c, x^*) - \mathbb{E}\left[\sum_{t=1}^{T} f(c_t, x_t)\right], \tag{13}$$

$$\mathcal{V}_{ave}(T) := \mathbb{E}\left[\sum_{t=1}^{T} g(c_t, x_t)\right], \tag{14}$$

$$\mathcal{V}_{tail}(T) := \mathbb{E}\left[\sum_{t=1}^{T} G^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}\right]. \tag{15}$$

Note (15) is a strict metric dedicated to latency-sensitive (priority) requests because a small cumulative violation implies satisfactory tail latency for almost every priority request. Our goal is to show POBO achieves sublinear regret and violation, i.e.,

$$\lim_{T\to\infty} \mathcal{R}(T)/T = 0,\quad \lim_{T\to\infty} \mathcal{V}_{ave}(T)/T = 0,\quad \text{and}\quad \lim_{T\to\infty} \mathcal{V}_{tail}(T)/T = 0.$$

To state our results, we first introduce the following assumptions.

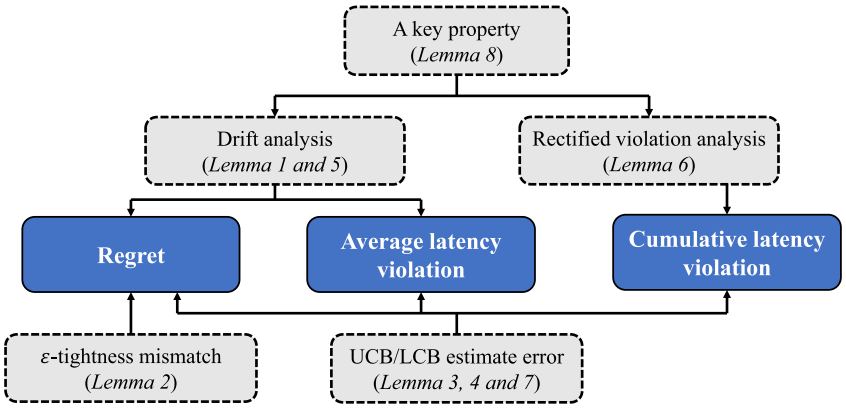


Fig. 4. Proof sketch of Theorem 1.

Assumption 1. Let $\|\cdot\|_k$ denote the RKHS norm associated with a kernel k. For the reward function f, we assume that $\|f\|_{k^f} \leq B_f$ and $k^f(c,x,x) \leq 1$ for any $x \in \mathcal{X}$. For the constraint function g, we assume $\|g\|_{k^g} \leq B_g$ and $k^g(c,x,x) \leq 1$ for any $x \in \mathcal{X}$. For the constraint function G, we assume $\|G\|_{k^G} \leq B_G$ and $k^G(c,x,x) \leq 1$ for any $x \in \mathcal{X}$.

Assumption 2. The noise η_t is i.i.d. R_f-sub-Gaussian, the noise ξ_t is i.i.d. R_g-sub-Gaussian and the noise τ_t is i.i.d. R_G-sub-Gaussian. The noise η_t, ξ_t, and τ_t are bounded by constants N_f, N_g, and N_G, respectively.

Assumption 3. There is a constant $\delta > 0$ such that there exists a probability distribution π_0 that satisfies $\int_{x\in\mathcal{X}} g(c,x)\pi_0(x)dx \leq -\delta$, $\forall c \in C$ and $\int_{x\in\mathcal{X}} G(c,x)\pi_0(x)dx \leq 0$, $\forall c \in C_p$. We assume $\delta \leq 1$.

Assumption 1 imposes the continuous and bounded condition on the black-box functions to guarantee they are learnable, which is typically mild and usually true in the microservice system as shown in Fig. 1(a). Assumption 2 is a common assumption for the noisy observation, and the noise is assumed bounded for the simplicity of presentation. Assumption 3 is standard Slater's condition in the optimization literature, which is used to derive an upper bound of violation $\mathcal{V}_{ave}(T)$ for the system-wide latency. This assumption is necessary because it requires a feasible configuration to strictly satisfy the latency constraints, which reflects the reality of the practical system since the resource configuration optimization could be impossible without the assumption. Moreover we define the parameters of GPs $\beta_t^f = B_f + R_f\sqrt{2(\gamma_t^f + 1 + \ln(3/p))}$, $\beta_t^g = B_g + R_g\sqrt{2(\gamma_t^g + 1 + \ln(3/p))}$ and $\beta_t^g = B_G + R_G\sqrt{2(\gamma_t^G + 1 + \ln(3/p))}$ with $p \in (0,1)$.

Next, we present our main results of the POBO algorithm in Theorem 1.

Theorem 1. *Under Assumptions 1–3, POBO achieves the following regret and constraint violations bound:*

$$\mathcal{R}(T) = O(\gamma_T\sqrt{T}), \quad \mathcal{V}_{ave}(T) = O(1), \quad \mathcal{V}_{tail}(T) = O(\gamma_T\sqrt{T}),$$

where $\gamma_T = max(\gamma_T^f, \gamma_T^g, \gamma_T^G)$.

The theorem shows POBO achieves $O(\gamma_T\sqrt{T})$ regret and "the best of two worlds" in constraint violation: $O(1)$ average violation as in [14] and $O(\gamma_T\sqrt{T})$ cumulative violation as in [14]. The constant average violation indicates that the resource configuration policy $x_1, x_2, \ldots, x_{T-1}, x_T$ ensures the received requests can be completed under SLA over periods T. Meanwhile, $O(\gamma_T\sqrt{T})$ cumulative violation indicates that tail latency requirements are strictly satisfied for almost every latency-sensitive request. Next, we prove the results in Theorem 1, and for a better picture, we first provide a roadmap in Fig. 4.

Remark 1. Recall that the definition $G^+(c_t, x_t)$ denotes the violation of tail latency for the request c_t with the configuration x_t. The result of $\mathcal{V}_{tail}(T) = O(\sqrt{T})$ implies the violation per request is $O(1/\sqrt{T})$. In other words, it means the tail latency requirement can always be satisfied under our algorithm for every request with a large T.

4.1. Regret bound

To prove the regret in Theorem 1, we introduce a stronger baseline than x^* in Section 2. Specifically, we define π as a distribution over the action set $\mathcal{X}$, and let π_* be the optimal solution to the following offline problem:

$$\max_{\pi} \mathbb{E}\left[\int_{x\in\mathcal{X}} f(c,x)\pi(x)dx\right] \tag{16}$$

$$\text{s.t. } \mathbb{E}\left[\int_{x\in\mathcal{X}} g(c,x)\pi(x)dx\right] \le 0, \tag{17}$$

$$\int_{x\in\mathcal{X}} G^{+}(c,x)\pi(x)dx = 0,\ \forall c \in C_p. \tag{18}$$

It is straightforward to have the following inequality

$$\mathcal{R}(T) \le \mathcal{R}_{+}(T) := \mathbb{E}\left[T\int_{x\in\mathcal{X}} f(c,x)\pi_{*}(x)dx - \sum_{t=1}^{T} f(c_t,x_t)\right]. \tag{19}$$

Further, we consider a tight problem corresponding to (16)–(18) with the tightness constant $0 \le \varepsilon \le \delta$ that

$$\max_{\pi}\ \mathbb{E}\left[\int_{x\in\mathcal{X}} f(c,x)\pi(x)dx\right] \tag{20}$$

$$\text{s.t. } \mathbb{E}\left[\int_{x\in\mathcal{X}} g(c,x)\pi(x)dx\right] + \varepsilon \le 0, \tag{21}$$

$$\int_{x\in\mathcal{X}} G^{+}(c,x)\pi(x)dx = 0,\ \forall c \in C_p. \tag{22}$$

Let $\pi_{*}^{\varepsilon_t}$ be the optimal solution to the above tight problem (20)–(22) with $\varepsilon = \varepsilon_t$. We decompose the cumulative regret in the following:

$$\begin{aligned}
\mathcal{R}_{+}(T) =\ & \mathbb{E}\left[T\int_{x\in\mathcal{X}} f(c,x)\pi_{*}(x)dx - \sum_{t=1}^{T} f(c_t,x_t)\right] \\
=\ & \mathbb{E}\left[\sum_{t=1}^{T}\int_{x\in\mathcal{X}} f(c_t,x)\pi_{*}(x)dx - \int_{x\in\mathcal{X}} f(c_t,x)\pi_{*}^{\varepsilon_t}(x)dx\right] && (23)\\
& + \mathbb{E}\left[\sum_{t=1}^{T}\int_{x\in\mathcal{X}} (f(c_t,x) - \hat{f}_t(c_t,x))\pi_{*}^{\varepsilon_t}(x)dx\right] && (24)\\
& + \mathbb{E}\left[\sum_{t=1}^{T}\int_{x\in\mathcal{X}} \hat{f}_t(c_t,x)\pi_{*}^{\varepsilon_t}(x)dx - \hat{f}_t(c_t,x_t)\right] && (25)\\
& + \mathbb{E}\left[\sum_{t=1} \hat{f}_t(c_t,x_t) - f(c_t,x_t)\right], && (26)
\end{aligned}$$

The term (23) is on the difference between the optimal offline problem and its $\epsilon-$tightness version. The terms (24) and (26) are on the learning errors of GP-UCB. The term (25) is the most challenging one, which can be established by the key property in Lemma 8. We then present a sequence of lemmas to bound the terms (23)–(26), respectively. The proof of all these lemmas can be found in Appendix.

Lemma 1. *Under POBO, we have*

$$\mathbb{E}\left[\sum_{t=1}^{T}\int_{x\in\mathcal{X}} \hat{f}_t(c_t,x)\pi_{*}^{\varepsilon_t}(x)dx - \hat{f}_t(c_t,x_t)\right] = O\left(\sum_{t=1}^{T}\frac{1}{V_t}\right).$$

Lemma 2. *Under Assumptions 1–3, we can bound the difference between the baseline optimization problem and its tightened version:*

$$\mathbb{E}\left[\sum_{t=1}^{T}\int_{x\in\mathcal{X}} f(c_t,x)\pi_{*}(x)dx - \int_{x\in\mathcal{X}} f(c_t,x)\pi_{*}^{\varepsilon_t}(x)dx\right] = O\left(\sum_{t=1}^{T}\varepsilon_t\right)$$

Lemma 3. *Under POBO, the GP-UCB estimator for reward f satisfies that*

$$\mathbb{E}\left[\sum_{t=1} \hat{f}_t(c_t,x_t) - f(c_t,x_t)\right] = O\left(\gamma_T^{f}\sqrt{T}\right),$$

$$\mathbb{E}\left[\sum_{t=1}^{T}\int_{x\in\mathcal{X}} (f(c_t,x) - \hat{f}_t(c_t,x))\pi_{*}^{\varepsilon_t}(x)dx\right] = O(1).$$

Given the above lemmas and recalling the values of learning rates V_t and ε_t, we have:

$$\begin{aligned}
\mathcal{R}_{+}(T) =\ & O\left(\sum_{t=1}^{T}\frac{1}{V_t} + \sum_{t=1}^{T}\varepsilon_t + \gamma_T\sqrt{T} + 1\right), \\
=\ & O\left(\sqrt{T} + \gamma_T\sqrt{T} + 1\right)
\end{aligned} \tag{27}$$

where $\gamma_T = max(\gamma_T^{f}, \gamma_T^{g}, \gamma_T^{G})$, which completes the proof of regret bound in Theorem 1.

4.2. Average latency violation bound

Then, we establish the average latency violation $\mathcal{V}_{ave}(T)$ by bounding Q_{T+1}. According to the update rule of Q_t in Eq. (11), we have

$$Q_{t+1} \geq Q_t + \check{g}_t(c_t, x_t) + \epsilon_t.$$

It implies that

$$Q_{T+1} \geq \sum_{t=1}^{T} \check{g}_t(c_t, x_t) + \sum_{t=1}^{T} \epsilon_t.$$

Therefore, we have

$$\begin{aligned}\mathcal{V}_{ave}(T) := \mathbb{E}\left[\sum_{t=1}^{T} g(c_t, x_t)\right] =& \mathbb{E}\left[\sum_{t=1}^{T} (g(c_t, x_t) - \check{g}_t(c_t, x_t))\right] + \mathbb{E}\left[\sum_{t=1}^{T} \check{g}_t(c_t, x_t)\right] \\ \leq& \mathbb{E}\left[\sum_{t=1}^{T} (g(c_t, x_t) - \check{g}_t(c_t, x_t))\right] + \mathbb{E}[Q_{T+1}] - \sum_{t=1}^{T} \epsilon_t, \end{aligned} \tag{28}$$

The first term in (28) is on the learning errors of GP-LCB, which can be bounded similarly as in the regret analysis.

Lemma 4. *Under POBO, the GP-LCB estimator for constraint g satisfies that*

$$\mathbb{E}\left[\sum_{t=1}^{T} g_t(c_t, x_t) - \check{g}(c_t, x_t)\right] = O(\gamma_T^g \sqrt{T}),$$

For the virtual queue Q_{t+1}, we use the Lyapunov drift analysis by the key property in Lemma 8 and establish the following lemma.

Lemma 5. *Under POBO, for $t \geq T'$ we have*

$$\begin{aligned}\mathbb{E}[Q_t] \leq& \frac{12(B_g+1)^2}{\delta} \log\left(\frac{8(B_g+1)}{\delta}\right) + (B_g+1) \\ &+ \frac{4(2B_f V_t + (B_g^2 + \epsilon_t^2))}{\delta} + B_g T' + \sum_{t=1}^{T'} \epsilon_t,\end{aligned}$$

where T' is the first period that satisfies $\epsilon_{T'} \leq \frac{\delta}{2}$.

Since T' is relatively small compared to the value of ϵ_t, we prove that for any time violation by carefully choosing $V_t = \frac{\delta\sqrt{t}}{8B_f}$ and $\epsilon_t = \frac{6\beta_T^g \sqrt{\gamma_T^g + 2}}{\sqrt{t}}$ such that

$$\mathcal{V}_{ave}(T) \leq \mathbb{E}\left[\sum_{t=1}^{T} (g(c_t, x_t) - \check{g}_t(c_t, x_t))\right] + \mathbb{E}[Q_{T+1}] - \sum_{t=1}^{T} \epsilon_t = O\left(1 + V_t - \sum_{t=1}^{T} \epsilon_t\right) = O(1).$$

This proves the constant average violation for the system-level latency in Theorem 1. From the regret analysis in (27) and the violation analysis above, we observe that the appropriate choice of ϵ_t is the key to the trade-off between the regret and the average constraint violation. This idea has been widely used in online learning literature [19] to establish a tight bound for average constraint violation.

4.3. Tail latency violation bound

Finally, we study the cumulative violation for tail latency. We first decompose the violation $\mathcal{V}_{tail}(T)$ in the following

$$\begin{aligned}\mathcal{V}_{tail}(T) :=& \mathbb{E}\left[\sum_{t=1}^{T} G^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}\right] \\ \leq& \mathbb{E}\left[\sum_{t=1}^{T} \check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}\right] + \mathbb{E}\left[\sum_{t=1}^{T} (G(c_t, x_t) - \check{G}_t(c_t, x_t))^+ \mathbb{I}\{c_t \in C_p\}\right].\end{aligned} \tag{29}$$

The first term on the cumulative violation of GP-LCB estimator $\check{G}_t(c_t, x_t)$, which can be established by the key property in Lemma 8 with properly choosing the learning rate η_t in POBO.

Lemma 6. *Under POBO, we have*

$$\mathbb{E}\left[\sum_{t=1}^{T} \check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}\right] = O(\sqrt{T}).$$

To translate the total violation of $\check{G}_t(c_t, x_t)$ into that of the original constraint function $G_t(c_t, x_t)$, we quantify the estimation errors between G_t and $\check{G}_t$.

Lemma 7. *Under POBO, the GP-LCB estimator for tail latency constraint G satisfies that*

$$\mathbb{E}\left[\sum_{t=1}^{T}(G(c_t,x_t)-\check{G}_t(c_t,x_t))\mathbb{I}\{c_t\in C_p\}\right] \le 4\beta_T^G\sqrt{4(T+2)\gamma_T^G}.$$

Combining these two lemmas, and we have

$$\mathcal{V}_{tail}(T) = O(\beta_T^G\sqrt{\gamma_T^G T}),$$

which completes the proof of the tail latency violation bound in Theorem 1 by $\beta_T^G = O(\sqrt{\gamma_T^G})$.

4.4. A key property and sketch of proof

Lemmas 1, 5, and 6 are the keys to establishing the regret, average violation, and cumulative violation, respectively. We introduce a crucial property to bridge the regret and constraint violations based on the Lyapunov drift analysis [20] that integrates the penalized constraint function. Intuitively, the property implies an upper bound for "regret + cumulative violation + average violation" as a whole and provides a unified analysis for these lemmas. We define the Lyapunov function and its drift to be

$$L(t) = \frac{1}{2}Q_t^2 \quad \text{and} \quad \Delta(t) = L(t+1) - L(t).$$

Lemma 8 (*A Key Property*)**.** *Under POBO, for any policy π, the following inequality holds*

$$\begin{aligned}&\frac{\Delta(t)}{V_t} + \hat{Q}_t\check{G}_t^+(c_t,x_t)\mathbb{I}\{c_t\in C_p\} + \int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi(x)dx - \hat{f}_t(c_t,x_t)\\ \le\ &\frac{Q_t}{V_t}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi(x)dx + \hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^+(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi(x)dx + \frac{(B_g^2+\varepsilon_t^2)}{V_t}.\end{aligned} \tag{30}$$

Lemma 8 suggests "Lyapunov drift + instantaneous violation + instantaneous regret" is bounded by the constraints violation associated with a policy π, from which we can individually analyze the regret $\mathcal{R}(T)$, average violation $\mathcal{V}_{ave}(t)$, and cumulative violation $\mathcal{V}_{tail}(t)$ in Theorem 1. Next, we outline the key steps in proving Lemmas 1, 5, and 6 with the inequality (31) in Lemma 8 and the detailed proof can be found in Appendix. For a simple exposition, we consider a fixed learning rate $V_t = V = O(\sqrt{T}), \eta_t = T, \forall t \in [T]$.

- For Lemma 1 on the analysis of regret, we establish the instantaneous regret from (31)

$$\begin{aligned}\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi(x)dx - \hat{f}_t(c_t,x_t) \le &-\frac{\Delta(t)}{V} + \frac{Q_t}{V}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi(x)dx\\ &+ \hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^+(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi(x)dx + \frac{(B_g^2+\varepsilon_t^2)}{V},\end{aligned}$$

  which implies the regret $\mathcal{R}(T) = O(T/V) = O(\sqrt{T})$ by summing t from 1 to T, because the first term $\sum_{t=1}^{T} -\frac{\Delta(t)}{V} = \frac{L(1)-L(T)}{V} \le 0$ and $\pi(x)$ is a feasible policy such that the violation in the second and third terms are non-positive.
- For Lemma 5 on the analysis of average violation, we have the Lyapunov drift from (31)

$$\begin{aligned}\Delta(t) \le\ &V\hat{f}_t(c_t,x_t) - V\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi(x)dx + Q_t\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi(x)dx\\ &+ V\hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^+(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi(x)dx + (B_g^2+\varepsilon_t^2),\end{aligned}$$

  which implies a negative drift

$$\Delta(t) \le -\frac{\delta}{2}Q_t + O(V).$$

  by realizing $\pi(x)$ is a strict feasible policy to the problem (16)–(18) such that $\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi(x)dx \le -\delta/2$ and the fourth term is zero. Therefore, according to the classical Lyapunov theory, we have $\mathbb{E}[Q_t] = O(V/\delta)$.
- For Lemma 6 on the analysis of cumulative violation, we have the instantaneous violation from (31)

$$\begin{aligned}\check{G}_t^+(c_t,x_t)\mathbb{I}\{c_t\in C_p\} \le &-\frac{\Delta(t)}{V\hat{Q}_t} + \frac{\hat{f}_t(c_t,x_t) - \int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi(x)dx}{\hat{Q}_t} + \frac{Q_t}{V\hat{Q}_t}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi(x)dx\\ &+ \int_{x\in\mathcal{X}}\check{G}_t^+(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi(x)dx + \frac{(B_g^2+\varepsilon_t^2)}{V\hat{Q}_t}.\end{aligned}$$

  By carefully choosing the penalty factor η_t in $\hat{Q}_t$ and realizing $\pi(x)$ is a feasible policy such that its violation in the fourth term is zero, we establish $\sum_{t=1}^{T}\check{G}_t^+(c_t,x_t)\mathbb{I}\{c_t\in C_p\} = O(V) = O(\sqrt{T})$.

5. Experimental evaluation

In this section, we present our experimental evaluation. We have implemented a prototype of POBO and conducted the experiments on a container cluster on a host equipped with a 2.5 GHz Intel i7-11700 CPU along with 32 GB memory and runs 64-bit Ubuntu 18.04. We first introduce our implementation details, followed by evaluation methodology, and then discuss the experimental results. Lastly, we discuss POBO's scalability, generalizability, and kernel selection in the latency–resource function learning. The code to replicate our experiments is available at https://github.com/caohch-1/POBO.

5.1. Implementation

Containerization: We use the Docker Desktop [21] as the containerization engine to install the necessary software with their dependencies for each container. Docker ensures that the target image is pulled to deliver the unit service for each container within a deployment. We leverage the commonly used container orchestration platform Kubernetes [22] to deploy, scale, and manage microservice applications. When we set up a microservice application, Kubernetes interacts with Docker to create a pool of pods and their containers within each deployment. For example, for the login service, 30 pods are created with a specification on how to run the containers, and 30 containers of the pods are created with the login image pulled.

Resource manager: For the resource manager, we implement the pod number controller by leveraging the Kubernetes Python client [23]. In our experiment, each pod consists of only one container. Each pod is configured with a 0.03 CPU core and 150MB of memory. By adjusting the number of pods, we can change the amounts of resources based on the determined resource configuration.

Benchmarking microservice and workload generator: We use the open-source microservice benchmark Deathstar [24] to deploy the Hotel Reservation application. We use the commonly used HTTP benchmarking tool wrk2 [25] as the workload generator to send three types of requests, i.e., login, search, and recommendation, to the application. Wrk2 provides APIs for setting different thread numbers, HTTP connections, requests, and duration. We set the duration of each period to 30 s and the number of requests to be sent during each period to 800 by using the exposed APIs.

Requests tracing: We use Jaeger [18] to implement end-to-end distributed tracing. POBO collects tracing data by leveraging Jaeger's exposed RESTful APIs that are instrumented in the functions of sending and receiving requests in the benchmarking microservice application's source code. Typically, we do not need to instrument the functions manually. Instead, developers provide the tracing instrumentation for sending and receiving requests in most microservice applications for troubleshooting.

As mentioned in Section 3.2, we down-sample the tracing to reduce the runtime overhead. Jaeger supports the probabilistic sampler to trace instrumented functions at a user-defined sampling rate. In our experiment, we set the sampling rate to 50%, which means the instrumented application functions are traced with a probability of 50%.

5.2. Evaluation methodology

We present the evaluation metrics followed by the baseline methods and then introduce our experiment design.

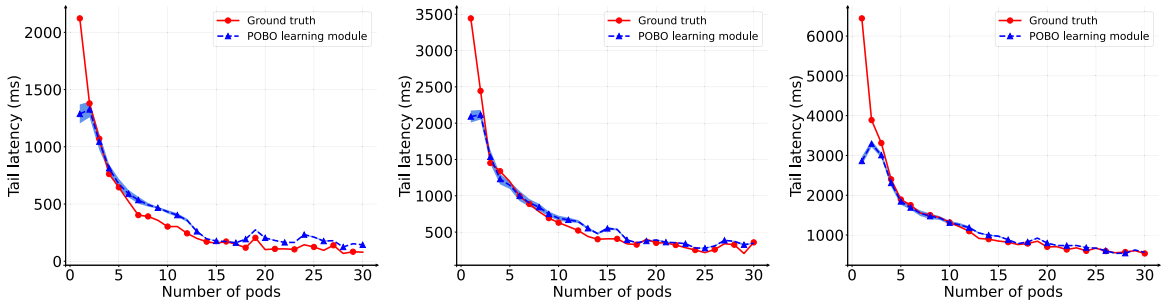
5.2.1. Evaluation metrics

We study the metrics of resource usage and service-level agreements (SLAs). For resource usage, we consider the average **number of pods** used to process a batch of requests. For SLAs, we consider two types of metrics on SLAs, which are commonly used in microservices [7]. The first one is **the percentage of violated requests**, where the violation means a request's response time exceeds the user-defined system-wide constraint. Ideally, the percentage of violated requests is low for a highly available microservice application. In the experiment, we set the threshold of system-wide response time to be 300, 400, and 800 ms for login, recommendation, and search, respectively. Those response time constraints vary in different environments, and our settings reflect the typical response times for three types of requests in our experimental environment. The second SLA metric is the **P90 tail latency** referring to the latency value below which 90% of requests' response time falls. In the experiment, we set the requirements of tail latency to be 400, 500, and 1000 ms for login, recommendation, and search, respectively. Still, the settings reflect the typical tail latencies for requests in our environment.

5.2.2. Baseline methods

We compare POBO with the Kubernetes Auto-scaler [26] and the state-of-the-art algorithm CKB [14].

Horizontal Pod Autoscaler (HPA): Kubernetes provides two types of auto-scalers, which are Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA). HPA is the scale-out method, meaning that we add more containers into the system, while VPA is the scale-up method, meaning that we add more resources to a particular container. In this paper, we adopt a scale-out approach, which is more popular in industrial cloud vendors. For scale-up approaches, containers must be restarted when the amount of resources is changed. Therefore, compared with scale-up approaches, scale-out approaches usually incur fewer downtimes, matching our design goal in the paper. Therefore, we compare POBO with the HPA method, which can automatically change the number of pods in response to the fluctuation of resource consumption. HPA continuously monitors the CPU or memory utilization and scales out/in the number of pods when the current metric value exceeds or falls below the target value. In particular, HPA provides a configurable parameter, i.e., the target resource utilization. HPA calculates the ratio α of the current resource utilization over the target resource utilization during runtime. HPA sets the newly adjusted pod number to the rounded value of ($\alpha\times$ the current pod number). In our experiment, we set the target CPU utilization to 60% by default.



(a) Login. (b) Recommendation. (c) Search.

Fig. 5. Tail latency–resource function learning for single-type requests.

Table 1
Converged results with single-type requests.

	Login			Recommendation		
	# of pods	% of violated req.	P90 latency (ms)	# of pods	% of violated req.	P90 latency (ms)
POBO	11.73	1.08	305.51	15.77	0.53	414.25
CKB	16.38	2.18	323.84	20.17	2.46	518.61
HPA	6.00	3.18	520.99	6.00	3.39	998.62

CKB: CKB [14] used Gaussian processes to learn the latency–resource functions and leverages primal–dual optimization to trade-off the resource usage and average violation instead of the cumulative violation. As CKB is designed to optimize the single-type request, we modify it by using a similar decomposition in this paper such that it applies to multiple-type requests. The detailed modification can be found in Appendix F.1.

As mentioned in the section of Introduction, RPOL [15] is another lightweight algorithm to solve safe BO. However, RPOL cannot be applied in our setting because RPOL is only designed for SLA on the tail latency, while our setting considers SLAs on both the average system latency and the tail latency for critical requests. Further discussion can be found in Appendix F.2.

5.2.3. Experiment design

We design two sets of experiments to verify the effectiveness of POBO and compare it with the baseline methods. For the first set of experiments, we evaluate POBO and other baselines by using the three metrics described in Section 5.2.1 when only single-type requests are received. To validate the performance of POBO's Learning module, we then compare the estimated latency–resource functions with the ground-truth functions. To obtain the ground-truth latency–resource functions, we measure the 800 arrived requests' tail latencies with changing numbers of pods. To reduce the inaccuracy incurred by the heavy tails of latencies, we repeat the experiment 10 times and show the average tail latency and the standard deviation at each pod number. Lastly, to verify the robustness and adaptivity of POBO in the setting of single-type requests, we perform a sensitivity study by varying the threshold of SLAs.

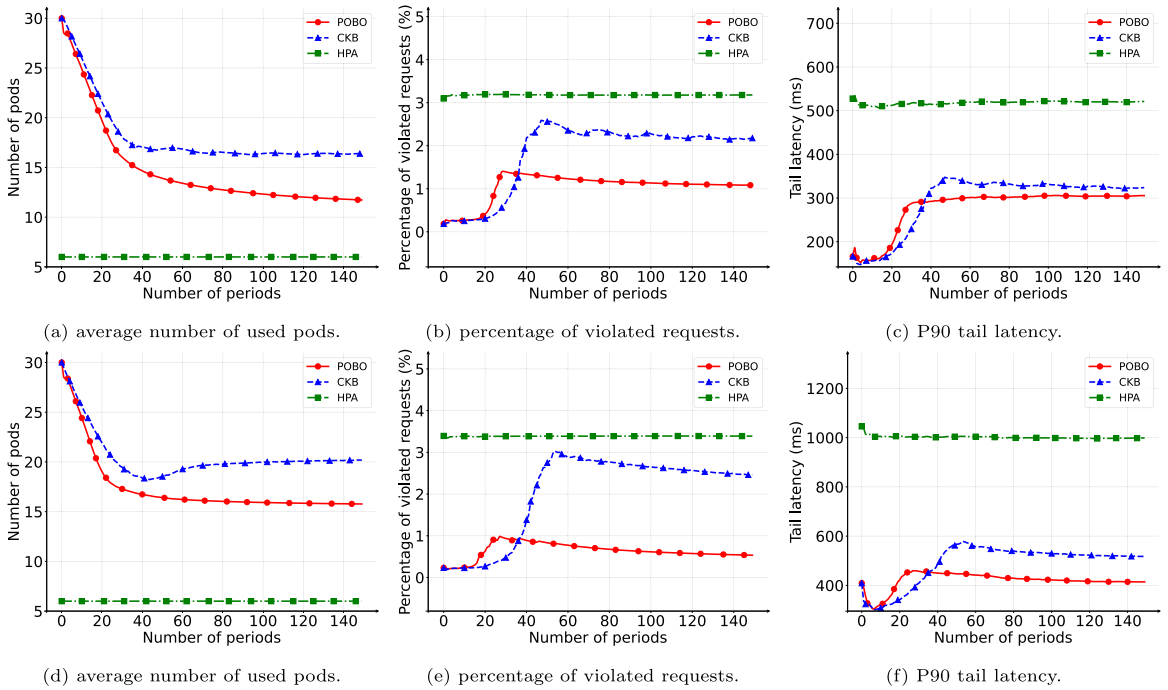
For the second set of experiments, we evaluate POBO and other baselines when multiple-type requests are received. We first consider a setting where multiple-type requests arrive according to a stationary process. In detail, we consider the requests of login, recommendation, and search generated from two distributions, i.e., [20%, 20%, 60%] and [40%, 40%, 20%], respectively. We then experiment on a slow-changing non-stationary distribution of incoming requests whose probabilistic distribution starts from [40%, 40%, 20%] and ends with [20%, 20%, 60%] in 500 periods, where the step size of the change is $[-0.04\%, -0.04\%, 0.08\%]$. Moreover, to evaluate the effectiveness of POBO in a dynamic environment, we conduct experiments by running a co-located reservation service simultaneously. Specifically, in each period, the arrival rate of the requests sent to the co-located service is randomly drawn from $\{0, 10, 20, 30\}$. The distribution of login, recommendation, and search requests is [33%, 33%, 33%]. Overall, for multiple-type requests, we investigate the robustness of POBO in the dynamic environment under non-stationary distributions of requests and dynamic interference.

5.3. Result analysis

In this section, we introduce the experimental results of resource configuration for single-type requests and multiple-type requests.

5.3.1. Warm-up: Resource configuration for single-type requests

Latency–resource function learning: GP model is a powerful framework for learning black-box functions. Fig. 5 shows the latency–resource function learned by the Gaussian process (GP) model, where we maintain individual GP models for each black-box function. The mean of the GP model predicts the black-box function, while the variance denotes the level of uncertainty. In Figs. 5(a)–5(c), the blue lines plot the mean of the GP models for P90 tail latency, while the light-shaded areas indicate the corresponding



(a) average number of used pods. (b) percentage of violated requests. (c) P90 tail latency.

(d) average number of used pods. (e) percentage of violated requests. (f) P90 tail latency.

Fig. 6. Warm-up experiment with single-type requests: login (above) and recommendation (below).

Table 2
Converged results under different stationary distributions of requests.

	Requests' distribution (0.2, 0.2, 0.6)			Requests' distribution (0.4, 0.4, 0.2)		
	# of pods	% of violated req.	P90 latency (ms)	# of pods	% of violated req.	P90 latency (ms)
POBO	10.23	2.28	58.11	10.36	2.10	122.93
CKB	11.10	2.38	69.21	12.36	2.38	146.94
HPA	6.00	4.17	100.72	6.00	3.57	211.31

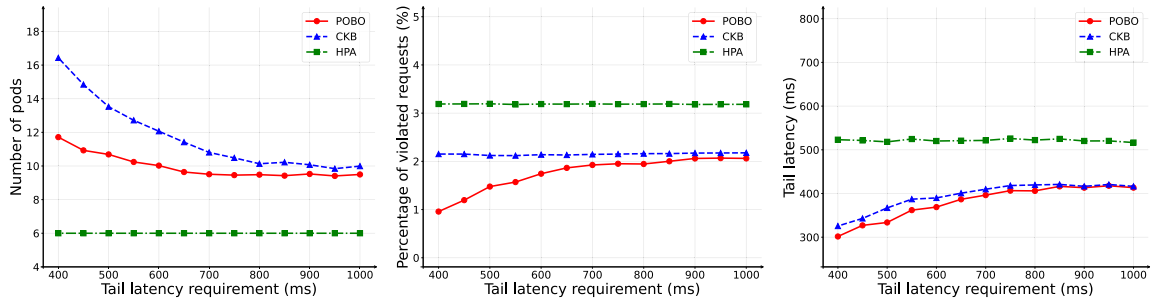
standard deviation for login, recommendation, and search requests, respectively. The results demonstrate that the function learned by POBO closely approximates the ground truth. It is also interesting to observe that the variance of the GP models tends to be larger when the number of pods is small. This observation is intuitive because POBO explores more frequently in configurations with a larger number of pods, where it is more likely to satisfy the P90 tail latency constraints.

Resource usage and SLAs: Figs. 6(a)–6(c) and 6(d)–6(f) plot the average number of pods, percentage of violated requests, and tail latencies under HPA, CKB, and POBO for the login and recommendation requests, respectively. It is shown that HPA suffers from a high percentage of violated requests and P90 tail latency as it determines the configuration based on the CPU or memory consumption. POBO outperforms CKB w.r.t average pod usage, the percentage of violated requests, and P90 tail latency. The converged results are summarized in Table 1. For the login and recommendation requests, POBO has a lower average pod usage with gaps of 4.65 and 4.40; a lower percentage of violated requests with gaps of 1.09% and 1.93%; and a better P90 latency with gaps of 18.34 and 104.36 ms. We also observe similar results for the search requests in Appendix F.4. These results demonstrate that POBO can efficiently find the optimal configuration of single-requests setting and consume fewer resources with minimal SLA violations.

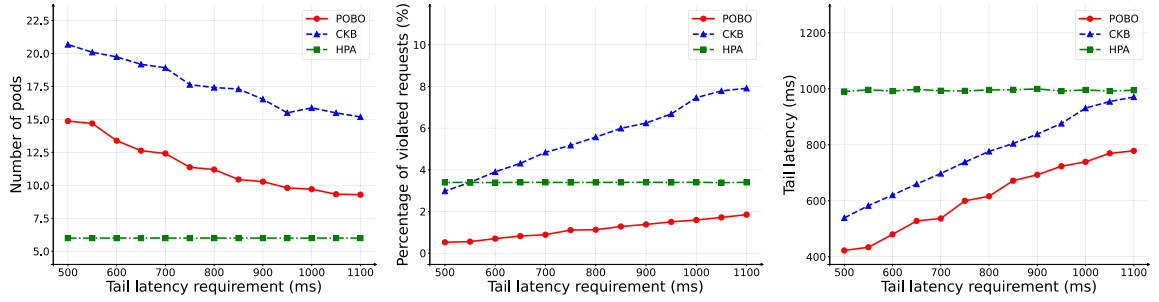
Sensitivity study: Fig. 7 plots the performance for the login and recommendation requests with the tail latency requirements from the strict 400 milliseconds to the loose 1000 milliseconds. It is expected that HPA is unaware of SLAs and has a fixed configuration. For the percentage of violated requests and tail latency, POBO outperforms CKB significantly when the threshold is strict. These results verify that POBO is robust and adaptive to the varying SLA requirement on tail latency.

5.3.2. Resource configuration for multiple-type requests

Stationary requests: We consider the requests of login, recommendation, and search generated from two distributions, i.e., [20%, 20%, 60%] and [40%, 40%, 20%], respectively. Figs. 8(a)–8(c) and 8(d)–8(f) plot the performance under HPA, CKB, and POBO for the two distributions, respectively. It is shown that HPA again suffers from a high percentage of violated requests and P90 tail latency. POBO outperforms CKB w.r.t average pod usage, the percentage of violated requests, and P90 tail latency for both distributions. The converged results are summarized in Table 2. For the two distributions, POBO has a lower average pod usage

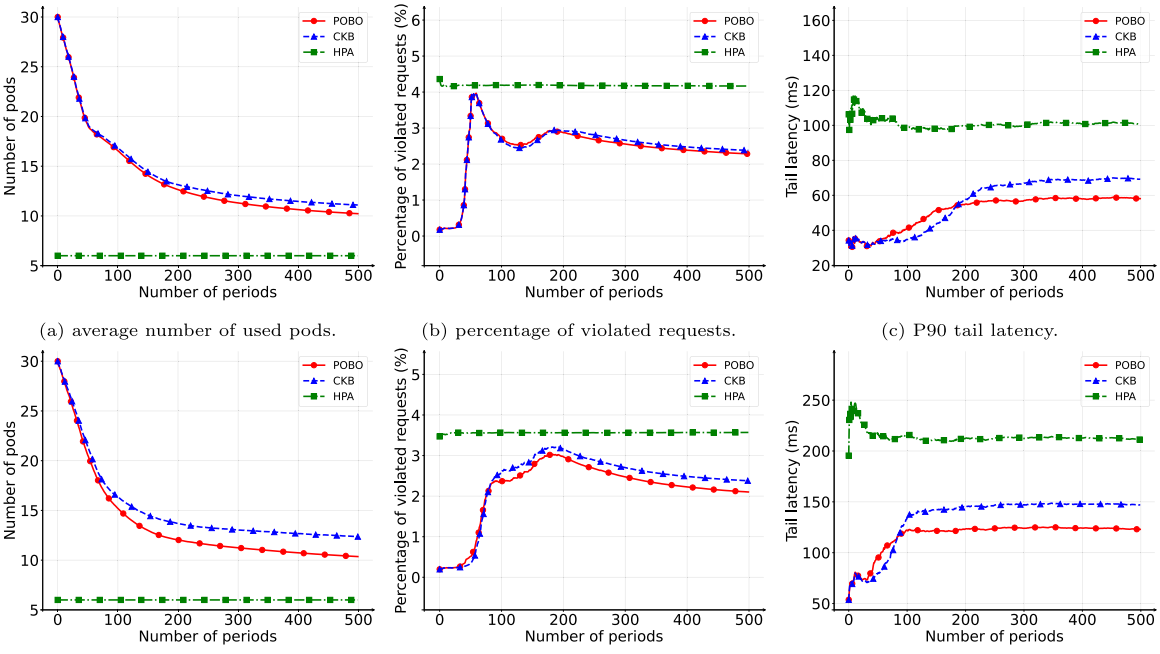


(a) Login - the average number of used pods in each period.
(b) Login - the percentage of violated requests in each period.
(c) Login - P90 tail latency of requests in each period.



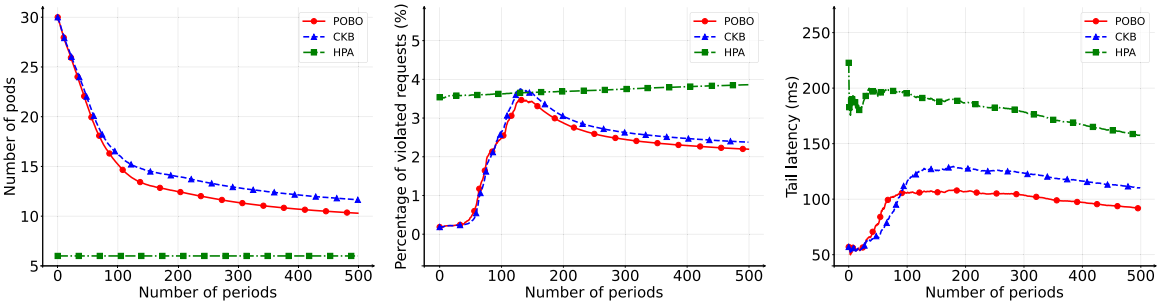
(d) Recommendation - the average number of used pods in each period.
(e) Recommendation- the percentage of violated requests in each period.
(f) Recommendation - P90 tail latency of requests in each period.

Fig. 7. Sensitivity study with single-type requests.



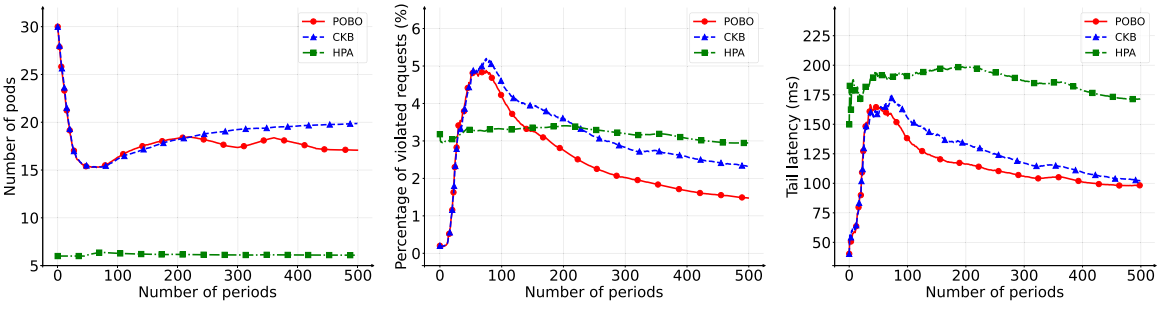
(a) average number of used pods.
(b) percentage of violated requests.
(c) P90 tail latency.
(d) average number of used pods.
(e) percentage of violated requests.
(f) P90 tail latency.

Fig. 8. The experimental results two different stationary distributions of requests: $[0.2, 0.2, 0.6]$ (above) and $[0.4, 0.4, 0.2]$ (below).



(a) average number of pods. (b) percentage of violated requests. (c) P90 tail latency.

Fig. 9. The experimental results under a non-stationary traffic pattern with a fixed number of arrivals.



(a) Average number of used pods. (b) Percentage of violated requests. (c) P90 tail latency.

Fig. 10. The experimental results under a fixed arrival distribution with varying arrivals in [27].

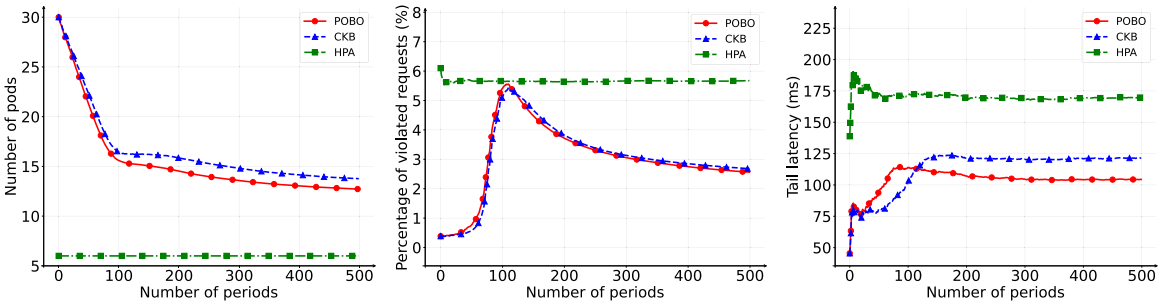
with gaps of 0.87 and 2.00; a lower percentage of violated requests with the gaps 0.10% and 0.28%; and a better P90 latency with the gaps 11.10 and 24.01 ms. These results demonstrate that POBO can quickly learn the workload statistics and converge to a safe and optimal solution even without any initial knowledge of the environment.

Non-stationary requests: We consider two types of non-stationary traffic patterns: (1) a non-stationary arrival distribution with a fixed total number of arrivals, and (2) a fixed arrival distribution with a varying number of arrivals. For the first traffic pattern, the probabilistic distribution starts from [40%, 40%, 20%] and ends with [20%, 20%, 60%] in 500 periods, where the step size of change is $[-0.04\%, -0.04\%, 0.08\%]$. Fig. 9 shows that POBO still performs best even under non-stationary distributions. We focus on comparing POBO with CKB as HPA fails to respond to the non-stationary environment. Fig. 9 shows the average pod usage of POBO is 1.35 lower than CKB; the percentage of violated requests of POBO is 0.18 lower than CKB; the P90 tail latency of POBO is 18.26 ms lower than CKB. For the second traffic pattern, we test POBO with a daily workload pattern extracted from real-world data collected in the production environment [27]. As shown in Fig. F.19 in Appendix F.6, the traffic is highly non-stationary (including original and interpolated data points). We plot the results in Fig. 10, and it is shown that the average pod usage of POBO is 2.78 lower than CKB; the percentage of violated requests of POBO is 0.85 lower than CKB; the P90 tail latency of POBO is 4.05 ms lower than CKB. The experiments further demonstrate that POBO performs well even under non-stationary arrival requests and that POBO is adaptive to the most effective policy with minimal resource usage and SLA violations.

Dynamic interference caused by co-located services. Fig. 11 shows that POBO is robust to dynamic interference and performs better than CKB and HPA. Similarly, we compare POBO with CKB in detail. As shown in Fig. 11(a), the average pod usage of POBO is 1.16 lower than CKB. For the percentage of violated requests, as shown in Fig. 11(b), POBO is 0.06 lower than CKB. For P90 tail latency, as shown in Fig. 11(c), POBO is 15.39 ms lower than CKB. From these results, we observe that the online learning and decision design in POBO are adaptive and robust in a dynamic environment. POBO can achieve a smaller tail latency with the penalty-based design for critical metrics.

5.4. Extensions

Scalability of POBO: POBO can be extended easily when pods of one identical service are implemented on one server. For example, the pods of the login service are all implemented on Server 1, the pods of the search service are all implemented on Server 2, etc. In this case, POBO can learn the latency–resource function well and give the optimal resource configuration without modification. However, when pods of one particular service are implemented on different servers, the latency–resource function cannot be learned since the requests' latencies are not solely based on the number of pods. For example, the login service is



(a) average number of pods. (b) percentage of violated requests. (c) P90 tail latency.

Fig. 11. The experimental results under dynamic interference caused by co-located services.

implemented on Servers 1 and 2. The workload generator and Server 1 are connected by the LAN network, and the workload generator and Server 2 are connected by the WAN network. In this case, it is essential to consider various network conditions among multiple servers when learning login requests' latencies.

Generalizability of POBO: POBO is a general framework and could be adaptive to other scenarios with slight modification because it learns the black-box functions (e.g., the latency–resource functions) from scratch and adjusts the resource configuration in real-time. For example, when the workloads are heavy-tailed, or the feedback suffers from large delay, we only need to customize the design of learning modules in POBO to tackle the specific challenges.

Kernel selection for POBO: We use a Squared Exponential (SE) kernel for POBO based on the belief of inherent "smoothness" in our system (i.e., the latency–resource relation is smooth). POBO is also flexible to incorporate other types of kernel or dynamic strategies of kernel selection to enhance the algorithm in the dynamic environment.

6. Conclusion

In this paper, we study the problem of microservice resource management and propose POBO, a novel framework that integrates pessimistic and optimistic learning with primal–dual and penalty-based optimization. We prove that POBO achieves the sublinear regret and average and cumulative violations. This implies that POBO can maximize resource efficiency while satisfying SLAs on the system-wide latency and tail latency even without the knowledge of the environment. We implement POBO on a real-world microservice system. The experimental results demonstrate that it outperforms the state-of-the-art algorithm and is robust and adaptive to the non-stationary environment.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors would like to thank the anonymous reviewers and the shepherd for their valuable comments. The work was partly supported by the Shanghai Sailing Program 22YF1428600 and 22YF1428500, the National Nature Science Foundation of China under grant 62302305.

Appendix A. A key property

Lemma 8 (*A Key Property*). *Under POBO, for any policy π, the following inequality holds*

$$\begin{aligned}&\frac{\Delta(t)}{V_t}+\hat{Q}_t\check{G}_t^+(c_t,x_t)\mathbb{I}\{c_t\in C_p\}+\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi(x)dx-\hat{f}_t(c_t,x_t)\\ \leq\ &\frac{Q_t}{V_t}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi(x)dx+\hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^+(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi(x)dx+\frac{(B_g^2+\varepsilon_t^2)}{V_t}.\end{aligned}\tag{31}$$

Proof. Recall the decision Eq. (10) in POBO algorithm, x_t is chosen such that

$$\arg\max_{x\in\mathcal{X}} \hat{f}_t(c_t,x) - \frac{Q_t\check{g}_t(c_t,x)}{V_t} - \hat{Q}_t\check{G}_t^+(c_t,x)\mathbb{I}\{c_t\in C_p\}.$$

For any policy π, we have

$$\begin{aligned}&\frac{Q_t\check{g}_t(c_t,x_t)}{V_t} + \hat{Q}_t\check{G}_t^+(c_t,x_t)\mathbb{I}\{c_t\in C_p\} - \hat{f}_t(c_t,x_t)\\ \le& -\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi(x)dx + \frac{Q_t}{V_t}\int_{x\in\mathcal{X}}\check{g}_t(c_t,x)\pi(x)dx + \hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^+(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi(x)dx.\end{aligned}$$

Next, we associate the term of $Q_t\check{g}_t(c_t,x_t)$ with the Lyapunov drift. The Lyapunov drift is

$$\Delta(t) = L(t+1) - L(t) = \frac{1}{2}Q_{t+1}^2 - \frac{1}{2}Q_t^2.$$

Recall the update rule of Q_t in Eq. (11), we have

$$\begin{aligned}\Delta(t) &\le \frac{1}{2}(2Q_t + \check{g}_t(c_t,x_t) + \varepsilon_t)(\check{g}_t(c_t,x_t) + \varepsilon_t)\\ &\le Q_t(\check{g}_t(c_t,x_t) + \varepsilon_t) + \frac{1}{2}(\check{g}_t(c_t,x_t) + \varepsilon_t)^2\\ &\le Q_t(\check{g}_t(c_t,x_t) + \varepsilon_t) + (B_g^2 + \varepsilon_t^2),\end{aligned}$$

Rearranging the above inequality, we have

$$\Delta(t) - Q_t\varepsilon_t - (B_g^2 + \varepsilon_t^2) \le Q_t\check{g}_t(c_t,x_t).$$

Combine these results, we prove Lemma 8 as follows

$$\begin{aligned}&\frac{\Delta(t)}{V_t} + \hat{Q}_t\check{G}_t^+(c_t,x_t)\mathbb{I}\{c_t\in C_p\} - \hat{f}_t(c_t,x_t)\\ \le& -\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi(x)dx + \frac{Q_t}{V_t}\int_{x\in\mathcal{X}}\check{g}_t(c_t,x)\pi(x)dx + \hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^+(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi(x)dx\\ &+ \frac{Q_t\varepsilon_t}{V_t} + \frac{(B_g^2+\varepsilon_t^2)}{V_t}.\end{aligned}$$

Appendix B. GP-UCB/LCB estimation errors

The errors of GP-UCB/LCB estimation are well established in [14,15,17,28]. We introduce GP-UCB/LCB estimate errors in Lemma 9 and the accumulative deviation in Lemma 10. The proof of these two lemmas can be found in [17].

Lemma 9. *Under Assumptions 1 and 2, the following event $\mathcal{E}$ hold for any $x\in\mathcal{X}$ and all $t\in[T]$*

$$\begin{aligned}&0 \le \hat{f}_t(c_t,x) - f(c_t,x) \le 2\beta_t^f\sigma_t^f(c_t,x),\\ &0 \le g(c_t,x) - \check{g}_t(c_t,x) \le 2\beta_t^g\sigma_t^g(c_t,x),\\ &0 \le G(c_t,x) - \check{G}_t(c_t,x) \le 2\beta_t^G\sigma_t^G(c_t,x),\end{aligned}$$

with probability at least $1-p$ with $p\in(0,1)$.

Lemma 10. *Let $\{x_1,\dots,x_T\}$ be the collection of decisions chosen by POBO. The cumulative standard deviation can be bounded as follows:*

$$\begin{aligned}&\sum_{t=1}^{T}\beta_t^f\sigma_t^f(c_t,x_t) \le \beta_T^f\sqrt{4(T+2)\gamma_T^f},\\ &\sum_{t=1}^{T}\beta_t^g\sigma_t^g(c_t,x_t) \le \beta_T^g\sqrt{4(T+2)\gamma_T^g},\\ &\sum_{t=1}^{T}\beta_t^G\sigma_t^G(c_t,x_t) \le \beta_T^G\sqrt{4(T+2)\gamma_T^G}.\end{aligned}$$

Before we prove Lemmas 3, 4 and 7, we first define *the high probability event* that

$$\mathcal{E} = \begin{Bmatrix}0 \le \hat{f}_t(c_t,x) - f(c_t,x) \le 2\beta_t^f\sigma_t^f(c_t,x),\\ 0 \le g(c_t,x) - \check{g}_t(c_t,x) \le 2\beta_t^g\sigma_t^g(c_t,x),\\ 0 \le G(c_t,x) - \check{G}_t(c_t,x) \le 2\beta_t^G\sigma_t^G(c_t,x), \forall x,t.\end{Bmatrix}$$

and

$$\mathbb{P}(\mathcal{E}) \geq 1 - p.$$

Note the event $\mathcal{E}$ will be frequently used in the proof of Theorem 1. Now we are ready to prove Lemmas 3, 4 and 7, which are regarded as "over-estimation errors" and "under-estimation errors". We present them in a unified way in Lemma 11 and Lemma 12, respectively.

B.1. Proof of Lemmas 3, 4 and 7

Lemma 11 (*Over-Estimation Errors*). *Let $\{x_1, \ldots, x_T\}$ be the collection of decisions chosen by the algorithm. Under Algorithm POBO, the following inequalities hold for all $x \in \mathcal{X}$:*

$$\mathbb{E}\left[\sum_{t=1}^{T} \hat{f}_t(c_t, x_t) - f(c_t, x_t)\right] \leq 2B_f Tp + 2\beta_T^f \sqrt{4(T+2)\gamma_T^f},$$
$$\mathbb{E}\left[\sum_{t=1}^{T} g(c_t, x_t) - \check{g}_t(c_t, x_t)\right] \leq 2B_g Tp + 2\beta_T^g \sqrt{4(T+2)\gamma_T^g},$$
$$\mathbb{E}\left[\sum_{t=1}^{T} G(c_t, x_t) - \check{G}_t(c_t, x_t)\right] \leq 2B_G Tp + 2\beta_T^G \sqrt{4(T+2)\gamma_T^G}.$$

Proof. Combine Lemmas 9 and 10, we have

$$\begin{aligned}
\mathbb{E}\left[\sum_{t=1}^{T} \hat{f}_t(c_t, x_t) - f(c_t, x_t)\right] &= \mathbb{E}\left[\sum_{t=1}^{T} \hat{f}_t(c_t, x_t) - f(c_t, x_t)|\bar{\mathcal{E}}\right] + \mathbb{E}\left[\sum_{t=1}^{T} \hat{f}_t(c_t, x_t) - f(c_t, x_t)|\mathcal{E}\right] \\
&\leq 2B_f Tp\mathbb{P}(\bar{\mathcal{E}}) + \mathbb{E}\left[\sum_{t=1}^{T} \hat{f}_t(c_t, x_t) - f(c_t, x_t)|\mathcal{E}\right] \\
&\leq 2B_f Tp + 2\beta_T^f \sqrt{4(T+2)\gamma_T^f}
\end{aligned}$$

Lemma 12 (*Under-Estimation Errors*). *Let $\{x_1, \ldots, x_T\}$ be the collection of decisions chosen by the algorithm. Under Algorithm POBO, the following inequalities hold for all $x \in \mathcal{X}$:*

$$\mathbb{E}\left[\sum_{t=1}^{T} f(c_t, x) - \hat{f}_t(c_t, x)\right] \leq 2B_f Tp$$
$$\mathbb{E}\left[\sum_{t=1}^{T} \check{g}_t(c_t, x) - g(c_t, x)\right] \leq 2B_g Tp$$
$$\mathbb{E}\left[\sum_{t=1}^{T} \check{G}_t(c_t, x) - G(c_t, x)\right] \leq 2B_G Tp$$

Proof. Combining Lemmas 9 and 10, we have

$$\begin{aligned}
\mathbb{E}\left[\sum_{t=1}^{T} f(c_t, x) - \hat{f}_t(c_t, x)\right] &= \mathbb{E}\left[\sum_{t=1}^{T} f(c_t, x) - \hat{f}_t(c_t, x)|\bar{\mathcal{E}}\right] + \mathbb{E}\left[\sum_{t=1}^{T} f(c_t, x) - \hat{f}(c_t, x)|\mathcal{E}\right] \\
&\leq 2B_f T\mathbb{P}(\bar{\mathcal{E}}) + \mathbb{E}\left[\sum_{t=1}^{T} f(c_t, x) - \hat{f}(c_t, x)|\mathcal{E}\right] \\
&\leq 2B_f Tp.
\end{aligned}$$

Similarly, we prove the cumulative under-estimation errors for g and G.

Appendix C. Proof of regret bound in Theorem 1

Recall the regret decomposition in (23)–(24), we have

$$\begin{aligned}
\mathcal{R}_+(T) &= \mathbb{E}[T\int_{x\in\mathcal{X}} f(c, x)\pi_*(x)dx - \sum_{t=1}^{T} f(c_t, x_t)] \\
&= \underbrace{\mathbb{E}[\sum_{t=1}^{T}(\int_{x\in\mathcal{X}} f(c_t, x)\pi_*(x)dx - \int_{x\in\mathcal{X}} f(c_t, x)\pi_*^{\varepsilon_t}(x)dx)]}_{\varepsilon_t\text{-tight}}
\end{aligned}$$

$$+\underbrace{\mathbb{E}[\sum_{t=1}^{T}\int_{x\in\mathcal{X}}(f(c_t,x)-\hat{f}_t(c_t,x))\pi_*^{\varepsilon_t}(x)dx]}_{\text{reward mismatch}}$$
$$+\underbrace{\mathbb{E}[\sum_{t=1}^{T}(\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi_*^{\varepsilon_t}(x)dx-\hat{f}_t(c_t,x_t))]}_{\text{Lyapunov drift}}$$
$$+\underbrace{\mathbb{E}[\sum_{t=1}\hat{f}_t(c_t,x_t)-f(c_t,x_t)]}_{\text{reward mismatch}}.$$

The reward mismatch in Lemmas 11 and 12 has been proved in Appendix B.1. We need to prove the difference between the two policies in Lemma 2 and the Lyapunov drift analysis in Lemma 1 with the key property in Lemma 8.

C.1. Proof of Lemma 1

According to the key property in Lemma 8, we rearrange terms in (31) and have

$$\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi(x)dx-\hat{f}_t(c_t,x_t)\le-\frac{\Delta(t)}{V_t}+\frac{Q_t}{V_t}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi(x)dx + \hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^{+}(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi(x)dx+\frac{(B_g^2+\varepsilon_t^2)}{V_t}.$$

Let $H_t=[Q_t,\hat{Q}_t,\hat{f}_t,\check{g}_t,\check{G}_t]$ and $h=[Q,\hat{Q},\hat{f},\check{g},\check{G}]$. Let $\pi^{\varepsilon}=\pi_*^{\varepsilon_t}$ and we have

$$\mathbb{E}[\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi_*^{\varepsilon_t}(x)dx-\hat{f}_t(c_t,x_t)|H_t=h]\le-\mathbb{E}[\frac{\Delta(t)}{V_t}|H_t=h]+\frac{(B_g^2+\varepsilon_t^2)}{V_t} + \mathbb{E}[\frac{Q_t}{V_t}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi_*^{\varepsilon_t}(x)dx|H_t=h] \tag{C.1}$$
$$+\mathbb{E}[\hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^{+}(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi_*^{\varepsilon_t}(x)dx|H_t=h]. \tag{C.2}$$

We study the terms in (C.1) and (C.2) as follows:

- For (C.1), we have

$$\begin{aligned}&\mathbb{E}[\frac{Q_t}{V_t}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi_*^{\varepsilon_t}(x)dx|H_t=h]\\ =\ &\frac{Q}{V_t}\mathbb{E}[\int_{x\in\mathcal{X}}(g(c_t,x)+\varepsilon_t)\pi_*^{\varepsilon_t}(x)dx|H_t=h]+\mathbb{E}[\frac{Q_t}{V_t}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)-g(c_t,x))\pi_*^{\varepsilon_t}(x)dx|H_t=h]\\ \le\ &\mathbb{E}[\frac{Q_t}{V_t}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)-g(c_t,x))\pi_*^{\varepsilon_t}(x)dx|H_t=h]\end{aligned}$$

  where the inequality holds because the underlying arrival process $\mathbb{P}(c_t)$ is independent with the history H_t and $\pi_*^{\varepsilon_t}$ is a feasible solution to the problem (20)–(22) with $\varepsilon=\varepsilon_t$.

- For (C.2), we have

$$\begin{aligned}&\mathbb{E}[\hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^{+}(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi_*^{\varepsilon_t}(x)dx|H_t=h]\\ \le\ &\hat{Q}\mathbb{E}[\int_{x\in\mathcal{X}}G^{+}(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi_*^{\varepsilon_t}(x)dx|H_t=h]+\mathbb{E}[\hat{Q}_t\int_{x\in\mathcal{X}}(\check{G}_t(c_t,x)-G(c_t,x))^{+}\mathbb{I}\{c_t\in C_p\}\pi_*^{\varepsilon_t}(x)dx|H_t=h]\\ \le\ &\mathbb{E}[\hat{Q}_t\int_{x\in\mathcal{X}}(\check{G}_t(c_t,x)-G(c_t,x))^{+}\mathbb{I}\{c_t\in C_p\}\pi_*^{\varepsilon_t}(x)dx|H_t=h]\end{aligned}$$

  where the first inequality holds because $(G+\check{G}-G)^{+}\le G^{+}+(\check{G}-G)^{+}$ and $\pi_*^{\varepsilon_t}$ is a feasible solution to the problem (20)–(22) with $\varepsilon=\varepsilon_t$ such that $\int_{x\in\mathcal{X}}G^{+}(c_t,x)\pi_*^{\varepsilon_t}(x)dx=0,\forall c_t\in C_p$.

Therefore, we have

$$\begin{aligned}&\mathbb{E}[\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi_*^{\varepsilon_t}(x)dx-\hat{f}_t(c_t,x_t)|H_t=h]\\ \le&-\mathbb{E}[\frac{\Delta(t)}{V_t}|H_t=h]+\frac{(B_g^2+\varepsilon_t^2)}{V_t}+\mathbb{E}[\frac{Q_t}{V_t}\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)-g(c_t,x))\pi_*^{\varepsilon_t}(x)dx|H_t=h]\\ &+\mathbb{E}[\hat{Q}_t\int_{x\in\mathcal{X}}(\check{G}_t(c_t,x)-G(c_t,x))^{+}\mathbb{I}\{c_t\in C_p\}\pi_*^{\varepsilon_t}(x)dx|H_t=h].\end{aligned}$$

Taking expectations w.r.t. H_t on both sides and doing the telescope summation up to T, we have

$$\begin{aligned}
&\mathbb{E}\left[\sum_{t=1}^{T}(\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi_*^{\varepsilon_t}(x)dx-\hat{f}_t(c_t,x_t))\right]\\
\leq&-\mathbb{E}[\sum_{t=1}^{T}\frac{\Delta(t)}{V_t}]+\sum_{t=1}^{T}\frac{(B_g^2+\varepsilon_t^2)}{V_t}+\mathbb{E}[\sum_{t=1}^{T}\frac{Q_t}{V_t}\int_{x\in\mathcal{X}}(g(c_t,x)-\check{g}_t(c_t,x))\pi_*^{\varepsilon_t}(x)dx]\\
&+\mathbb{E}[\sum_{t=1}^{T}\hat{Q}_t\int_{x\in\mathcal{X}}(\check{G}_t(c_t,x)-G(c_t,x))^+\mathbb{I}\{c_t\in C_p\}\pi_*^{\varepsilon_t}(x)dx]\\
\leq&\mathbb{E}[\frac{L(1)}{V_1}]-\mathbb{E}[\frac{L(T+1)}{V_T}]+\sum_{t=1}^{T}\frac{(B_g^2+\varepsilon_t^2)}{V_t}+2pB_g\sum_{t=1}^{T}\frac{\mathbb{E}[Q_t]}{V_t}+2pB_G\sum_{t=1}^{T}\mathbb{P}(c_t\in C_p)\mathbb{E}[\hat{Q}_t]\\
\leq&\sum_{t=1}^{T}\frac{(B_g^2+\varepsilon_t^2)}{V_t}+2pB_g\sum_{t=1}^{T}\frac{(tB_g+\sum_{s=1}^{t}\varepsilon_s)}{V_t}+2\sum_{t=1}^{T}\mathbb{P}(c_t\in C_p)t(B_G+N_G)pB_G,
\end{aligned}\tag{C.3}$$

where the second inequality holds by Lemma 12; the last inequality holds comes from the fact that $L(1)=0$, $L(T+1)\geq 0$ and the bounded queue lengths

$$\begin{aligned}
&Q_{t+1}=\left(Q_t+\check{g}_t(c_t,x_t)+\varepsilon_t\right)^+\leq Q_t+B_g+\varepsilon_t\leq tB_g+\sum_{s=1}^{t}\varepsilon_s,\\
&\hat{Q}_t\leq\max(t(B_G+N_G),t)=t(B_g+N_g).
\end{aligned}$$

Finally, we choose $p=1/T^2$ and substitute ε_t and V_t in (C.3), which prove Lemma 1.

C.2. Proof of Lemma 2

Recall that π_* is the optimal solution to optimization problem (1)–(3). Under Assumption 3, there exists π_0 that satisfies

$$\begin{aligned}
&\mathbb{E}\left[\int_{x\in\mathcal{X}}g(c,x)\pi_0(x)dx\right]+\delta\leq 0,\\
&\int_{x\in\mathcal{X}}G^+(c,x)\pi_0(x)dx=0,\ \forall c\in C_p.
\end{aligned}$$

We mix π_* and π_0 to construct $\pi^{\varepsilon_t}=(1-\frac{\varepsilon_t}{\delta})\pi_*+\frac{\varepsilon_t}{\delta}\pi_0$ such that

$$\begin{aligned}
&\mathbb{E}\left[\int_{x\in\mathcal{X}}g(c,x)\pi^{\varepsilon_t}(x)dx\right]+\varepsilon_t\leq 0,\\
&\int_{x\in\mathcal{X}}G^+(c,x)\pi^{\varepsilon_t}(x)dx=0,\ \forall c\in C_p.
\end{aligned}$$

Thus π^{ε_t} is a feasible solution to the problem (20)–(22) with $\varepsilon=\varepsilon_t$. Recall $\pi_*^{\varepsilon_t}$ is the optimal solution to the same problem (20)–(22) with $\varepsilon=\varepsilon_t$, we have

$$\begin{aligned}
\mathbb{E}\left[\int_{x\in\mathcal{X}}(f(c,x)\pi_*(x)-f(c,x)\pi_*^{\varepsilon_t}(x))dx\right]&\leq\mathbb{E}\left[\int_{x\in\mathcal{X}}(f(c,x)\pi_*(x)-f(c,x)\pi^{\varepsilon_t}(x))dx\right]\\
&\leq\mathbb{E}\left[\int_{x\in\mathcal{X}}f(c,x)(\frac{\varepsilon_t}{\delta}\pi_*(x)-\frac{\varepsilon_t}{\delta}\pi_0(x))dx\right]\\
&\leq\frac{2B_f\varepsilon_t}{\delta},
\end{aligned}$$

where the first inequality holds since $\pi_*^{\varepsilon_t}$ is the optimal solution; the last inequality holds because $f(c,x)$ is bounded according to Assumption 1. The proof of Lemma 2 is completed.

C.3. Proving regret bound

With the decomposition and the above lemma, we have

$$\begin{aligned}
\mathcal{R}(T)\leq\mathcal{R}_+(T)\leq&\sum_{t=1}^{T}\frac{2B_f\varepsilon_t}{\delta}+2B_fTp\\
&+\sum_{t=1}^{T}\frac{(B_g^2+\varepsilon_t^2)}{V_t}+2pB_g\sum_{t=1}^{T}\frac{(tB_g+\sum_{s=1}^{t}\varepsilon_s)}{V_t}+2\sum_{t=1}^{T}\mathbb{P}(c_t\in C_p)t(B_g+N_g)pB_G\\
&+2B_fTp+2\beta_T^f\sqrt{4(T+2)\gamma_T^f}.
\end{aligned}$$

Recall $V_t = \frac{\delta\sqrt{t}}{8B_f}$, $\varepsilon_t = \frac{6\beta_T^g\sqrt{\gamma_T^g}+2}{\sqrt{t}}$ and $p = 1/T^2$, we have:

$$\sum_{t=1}^{T}\varepsilon_t \le (6\beta_T^g\sqrt{\gamma_t^g}+2)(2\sqrt{T}+1),\ \sum_{t=1}^{T}tp \le \frac{1}{2},\ 2B_fTp = \frac{2B_f}{T},$$
$$\sum_{t=1}^{T}\frac{(B_g^2+\varepsilon_t^2)}{V_t} \le \sum_{t=1}^{T}\frac{(B_g^2+k^2)}{V_t} \le \frac{16B_f(B_g^2+k^2)}{\delta}\sqrt{T},$$
$$p\sum_{t=1}^{T}\frac{(tB_g+\sum_{s=1}^{t}\varepsilon_s)}{V_t} \le p\sum_{t=1}^{T}\frac{8B_ft(B_g+k)}{\delta\sqrt{t}} \le \frac{16B_f(B_g+k)}{3\delta\sqrt{T}},$$

where $k = 6\beta_T^g\sqrt{\gamma_t^g}+2$. Finally, we have proved the regret bound in Theorem 1.

Appendix D. Proof of average latency violation bound in Theorem 1

Recall the decomposition of average latency violation given by (28), and we have

$$\mathcal{V}_{ave}(t) \le \mathbb{E}\left[\sum_{s=1}^{t}(g(c_s,x_s)-\check{g}_s(c_s,x_s))\right]+\mathbb{E}[Q_{t+1}]-\sum_{s=1}^{t}\varepsilon_s.$$

The estimation error in Lemma 4 has been proved in Appendix B.1. Next, we focus on proving Lemma 5.

D.1. Proof of Lemma 5

To establish the upper bound of Q_{t+1}, we present the following Lemma on Lyapunov drift analysis. The lemma is derived from [19] with a slight modification from [20,29], where the boundary (i.e., ϕ_t in the lemma) is allowed to be time-varying. The proof can be found in Lemma 11 in [19].

Lemma 13. *Let $\mathcal{E}$ be some event, $S(t)$ be a random process, $\Phi(t)$ be its Lyapunov function with $\Phi(0)=\Phi_0$ and $\Phi(S(t+1))-\Phi(S(t))$ be the Lyapunov drift. Given an increasing sequence $\{\varphi_t\}$, ρ v_{max} with $0\le\rho\le v_{max}$, if the expected drift satisfies the following $\mathbb{E}[\Delta(t)|S(t)=s,\mathcal{E}]$ satisfies the following condition:*

(i) There exist $\rho>0$ and $\varphi_t>0$ such that $\mathbb{E}[\Phi(S(t+1))-\Phi(S(t))|S(t)=s,\mathcal{E}]\le-\rho$ when $\Phi(s)\ge\varphi_t$,

(ii) $|\Phi(S(t+1))-\Phi(S(t))|\le v_{max}$ holds with probability one;

Then we have the following inequality with $\zeta=\frac{\rho}{v_{max}^2+v_{max}\rho/3}$

$$\mathbb{E}[e^{\zeta\Phi S(t)}|\mathcal{E}] \le e^{\zeta\Phi_0}+\frac{2e^{\zeta(v_{max}+\varphi_t)}}{\zeta\rho}.$$

We first prove the following lemma conditioned on this event and $\varepsilon_t\le\delta/2$ for any t.

Lemma 14. *Assume $\varepsilon_t\le\delta/2$, and let $\varphi_t=\frac{4(2B_fV_t+(B_g^2+\varepsilon_t^2))}{\delta}$. Under the POBO algorithm, we have*

(i) $\mathbb{E}[Q_{t+1}-Q_t|H_t=h,\mathcal{E}]\le-\frac{\delta}{4}$, when $Q_t\ge\varphi_t$;

(ii) $|Q_{t+1}-Q_t|\le B_g+1$ holds with probability one.

Proof. From Lemma 8, we immediately have for any policy π

$$\begin{aligned}\Delta(t) \le\ & V_t\hat{f}_t(c_t,x_t)-V_t\int_{x\in\mathcal{X}}\hat{f}_t(c_t,x)\pi(x)dx+Q_t\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi(x)dx\\ &+V_t\hat{Q}_t\int_{x\in\mathcal{X}}\check{G}_t^{+}(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi(x)dx+(B_g^2+\varepsilon_t^2).\end{aligned}$$

Given $H_t=h$ and the event $\mathcal{E}$, let $\pi=\pi^\epsilon$, we have

$$\begin{aligned}\mathbb{E}\left[\Delta(t)|H_t=h,\mathcal{E}\right] \le\ & 2B_fV_t+Q_t\mathbb{E}\left[\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi^\epsilon(x)dx|H_t=h,\mathcal{E}\right]\\ &+V_t\hat{Q}_t\mathbb{E}\left[\int_{x\in\mathcal{X}}\check{G}_t^{+}(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi^\epsilon(x)dx|H_t=h,\mathcal{E}\right]+(B_g^2+\varepsilon_t^2)\\ \le\ & 2B_fV_t+Q_t\mathbb{E}\left[\int_{x\in\mathcal{X}}(\check{g}_t(c_t,x)+\varepsilon_t)\pi^\epsilon(x)dx|H_t=h,\mathcal{E}\right]+(B_g^2+\varepsilon_t^2)\end{aligned}$$

where the first inequality holds due to the projector imposed on $\check{f}_t(c_t,x)$; the second inequality holds similarly as in (C.2) because π^ϵ is a feasible solution such that $\int_{x\in\mathcal{X}}G(c_t,x)\mathbb{I}\{c_t\in C_p\}\pi^\epsilon(x)dx=0$ and the $\check{G}_t^{+}(c_t,x)\le G(c_t,x)$ with a high probability. Moreover, as $\epsilon_t\le\delta/2$, there exists π^ϵ such that

$$\int_{x\in\mathcal{X}}\check{g}_t(c_t,x)\pi^\epsilon dx \le \int_{x\in\mathcal{X}}g(c_t,x)\pi^\epsilon dx \le -\delta.$$

Therefore, the expected drift is

$$\mathbb{E}\left[\Delta_t | H_t = h, \mathcal{E}\right] \leq -\frac{\delta}{2} Q_t + 2B_f V_t + (B_g^2 + \varepsilon_t^2).$$

Now we prove the conditions (i) and (ii).

- For condition (i), given that $H_t = h$ and $Q_t \geq \varphi_t = \frac{4(2B_f V_t + (B_g^2 + \varepsilon_t^2))}{\delta}$, the Lyapunov drift

$$\mathbb{E}\left[\frac{1}{2} Q_{t+1}^2 - \frac{1}{2} Q_t^2 | H_t = h, \mathcal{E}\right] \leq -\frac{\delta}{4} Q_t,$$

  Therefore, we have

$$\mathbb{E}\left[Q_{t+1}^2 | H_t = h, \mathcal{E}\right] \leq (Q_t - \frac{\delta}{2})^2,$$

  which implies

$$\mathbb{E}\left[Q_{t+1} - Q_t | H_t = h, \mathcal{E}\right] \leq -\frac{\delta}{2},$$

- For condition (ii), we have

$$|Q_{t+1} - Q_t| \leq B_g + \varepsilon_t \leq B_g + 1,$$

  where the last inequality holds since $\varepsilon_t \leq \delta/2 \leq 1$.

Proving Lemma 5**:** Let $\varphi_t = \frac{4(2B_f V_t + (B_g^2 + \varepsilon_t^2))}{\delta}$, $\rho = \delta/4$ and $v_{max} = B_g + 1$, we apply Lemma 13 to establish

$$\mathbb{E}[e^{\zeta Q_t} | \mathcal{E}] \leq e^{\zeta Q_{T'}} + \frac{2e^{\zeta(v_{max} + \varphi_t)}}{\zeta \rho},$$

where T' is the first round that satisfies $\varepsilon_{T'} \leq \delta/2$, we further show that

$$\begin{aligned}
\mathbb{E}[Q_t | \mathcal{E}] \leq & \frac{1}{\zeta} \log(e^{\zeta Q_{T'}} + \frac{2e^{\zeta(v_{max} + \varphi_t)}}{\zeta \rho}) \\
\leq & \frac{1}{\zeta} \log(e^{\zeta Q_{T'}} + \frac{8v_{max}^2}{3\rho^2} e^{\zeta(v_{max} + \varphi_t)}) \\
\leq & \frac{1}{\zeta} \log\left(\frac{11 v_{max}^2}{3\rho^2} e^{\zeta(Q_{T'} + v_{max} + \varphi_t)}\right) \\
\leq & \frac{3v_{max}^2}{\rho} \log(\frac{2v_{max}}{\rho}) + v_{max} + \varphi_t + Q_{T'} \\
\leq & \frac{12(B_g + 1)^2}{\delta} \log\left(\frac{8(B_g + 1)}{\delta}\right) + (B_g + 1) \\
& + \frac{4(2B_f V_t + (B_g^2 + \varepsilon_t^2))}{\delta} + B_g T' + \sum_{t=1}^{T'} \varepsilon_t.
\end{aligned}$$

Thus we can bound Q_t as follows:

$$\begin{aligned}
\mathbb{E}[Q_t] = & \mathbb{E}[Q_t | \mathcal{E}]\mathbb{P}(\mathcal{E}) + \mathbb{E}[Q_t | \bar{\mathcal{E}}]\mathbb{P}(\bar{\mathcal{E}}) \\
\leq & \mathbb{E}[Q_t | \mathcal{E}] + \sum_{s=1}^{t} (B_g + \varepsilon_s)\mathbb{P}(\bar{\mathcal{E}}) \\
\leq & \frac{12(B_g + 1)^2}{\delta} \log\left(\frac{8(B_g + 1)}{\delta}\right) + (B_g + 1) \\
& + \frac{4(2B_f V_t + (B_g^2 + \varepsilon_t^2))}{\delta} + B_g T' + \sum_{t=1}^{T'} \varepsilon_t + \sum_{s=1}^{t} (B_g + \varepsilon_s) p
\end{aligned}$$

D.2. Proving average latency violation bound

Finally, we prove the violation bound of average latency in Theorem 1. Recall decomposition given by (28):

$$\begin{aligned}
\mathcal{V}_{ave}(t) := \mathbb{E}\left[\sum_{s=1}^{t} g(c_s, x_s)\right] = & \mathbb{E}\left[\sum_{s=1}^{t} (g(c_s, x_s) - \check{g}_s(c_s, x_s))\right] + \mathbb{E}\left[\sum_{s=1}^{t} \check{g}_s(c_s, x_s)\right] \\
\leq & \mathbb{E}\left[\sum_{s=1}^{t} (g(c_s, x_s) - \check{g}_s(c_s, x_s))\right] + \mathbb{E}[Q_{t+1}] - \sum_{s=1}^{t} \varepsilon_s,
\end{aligned}$$

$$\leq 2B_g T' p + 2\beta_t^g \sqrt{4(T'+2)\gamma_t^g} + \mathbb{E}[Q_{t+1}] - \sum_{s=1}^{t} \varepsilon_s,$$

where the last inequality holds by Lemma 12. Consider the case that $t \leq T'$ and define $k = 6\beta_T^g \sqrt{\gamma_T^g} + 2$, from Eq. (11) we have

$$Q_{t+1} \leq tB_g + \sum_{s=1}^{t} \varepsilon_s \leq T' B_g + \sum_{s=1}^{T'} \varepsilon_s \leq T' B_g + 2k\sqrt{T'} + k.$$

Since $\sum_{s=1}^{t} \varepsilon_s \geq 0$, for anytime soft violation $\mathcal{V}_{ave}$ we have

$$\mathcal{V}_{ave}(t) \leq 2B_g T' p + 2\beta_t^g \sqrt{4(T'+2)\gamma_t^g} + T' B_g + 2k\sqrt{T'} + k.$$

Then for $t \geq T'$, we can still bound Q_{t+1} through Lemma 5 and get

$$\begin{aligned}\mathcal{V}_{ave}(t) \leq\ & 2B_g tp + 2\beta_t^g \sqrt{4(t+2)\gamma_t^g} + \frac{12(B_g+1)^2}{\delta} \log\left(\frac{8(B_g+1)}{\delta}\right) + (B_g+1) \\ & + \frac{4(2B_f V_{t+1} + (B_g^2 + \varepsilon_t^2))}{\delta} + B_g T' + \sum_{t=1}^{T'} \varepsilon_t + p \sum_{s=1}^{t+1} (B_g + \varepsilon_s) - \sum_{s=1}^{t} \varepsilon_s.\end{aligned}$$

Recall the values of V_t and ε_t, we establish $\mathcal{V}_{ave}(t)$ when $t \geq T'$, thereby completing the proof of the average latency violation as stipulated in Theorem 1.

Appendix E. Proof of cumulative tail latency violation in Theorem 1

Recall we decompose the cumulative violation in the following way

$$\mathbb{E}\left[\sum_{t=1}^{T} G^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}\right] = \mathbb{E}\left[\sum_{t=1}^{T} (G_t^+(c_t, x_t) + G(c_t, x_t) - \check{G}_t(c_t, x_t))^+ \mathbb{I}\{c_t \in C_p\}\right]$$

The estimation error in Lemma 7 has been proved in Appendix B.1, and we focus on proving Lemma 6.

E.1. Proof of Lemma 6

Let $\pi = \pi_*$ in Lemma 8. By rearranging the inequality, we have the cumulative violation

$$\begin{aligned}\hat{Q}_t \check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\} \leq & -\frac{\Delta(t)}{V_t} + \hat{f}_t(c_t, x_t) - \int_{x\in\mathcal{X}} \hat{f}_t(c_t, x)\pi_*(x)dx + \frac{Q_t}{V_t} \int_{x\in\mathcal{X}} (\check{g}_t(c_t, x) + \varepsilon_t)\pi_*(x)dx \\ & + \hat{Q}_t \int_{x\in\mathcal{X}} \check{G}_t^+(c_t, x)\mathbb{I}\{c_t \in C_p\}\pi_*(x)dx + \frac{(B_g^2 + \varepsilon_t^2)}{V_t}.\end{aligned}$$

In conjunction with $|\hat{f}| \leq B_f$, $|\check{g}| \leq B_g$ and $\hat{Q}_t \geq \eta_t$, we have

$$\begin{aligned}\check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\} \leq & \frac{-\Delta(t)}{\hat{Q}_t V_t} + \frac{2B_f}{\hat{Q}_t} + \frac{Q_t(B_g + \varepsilon_t)}{\hat{Q}_t V_t} + \frac{(B_g^2 + \varepsilon_t^2)}{\hat{Q}_t V_t} \\ & + \int_{x\in\mathcal{X}} \check{G}_t^+(c_t, x)\mathbb{I}\{c_t \in C_p\}\pi_*(x)dx \\ \leq & \frac{-\Delta(t)}{\eta_t V_t} + \frac{2B_f}{\eta_t} + \frac{Q_t(B_g + \varepsilon_t)}{\eta_t V_t} + \frac{(B_g^2 + \varepsilon_t^2)}{\eta_t V_t} \\ & + \int_{x\in\mathcal{X}} \check{G}_t^+(c_t, x)\mathbb{I}\{c_t \in C_p\}\pi_*(x)dx\end{aligned}$$

Under the event $\mathcal{E}$, we have

$$\check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\} \leq \frac{-\Delta(t)}{\eta_t V_t} + \frac{2B_f}{\eta_t} + \frac{Q_t(B_g + \varepsilon_t)}{\eta_t V_t} + \frac{(B_g^2 + \varepsilon_t^2)}{\eta_t V_t},$$

which implies

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T} \check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}|\mathcal{E}\right] \leq & \frac{L(1)}{V_1} - \frac{L(T)}{TV_T} + \sum_{t=1}^{T} \frac{2B_f}{t} + \sum_{t=1}^{T} \frac{B_g Q_t}{tV_t} + \sum_{t=1}^{T} \frac{Q_t \varepsilon_t}{tV_t} + \sum_{t=1}^{T} \frac{(B_g^2 + \varepsilon_t^2)}{tV_t} \\ \leq & \sum_{t=1}^{T} \frac{2B_f}{t} + \sum_{t=1}^{T} \frac{B_g Q_t}{tV_t} + \sum_{t=1}^{T} \frac{Q_t \varepsilon_t}{tV_t} + \sum_{t=1}^{T} \frac{(B_g^2 + \varepsilon_t^2)}{tV_t},\end{aligned}$$

where the last inequality comes from the definition of $L(t)$, we then recall the fact that

$$Q_t \le tB_g + \sum_{s=1}^{t} \epsilon_s \le tB_g + 2k\sqrt{t} + k \le tB_g + 3k\sqrt{t},$$

where $k = 6\beta_T^g\sqrt{\gamma_T^g} + 2$. We can immediately get the following bound

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T} \check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}|\mathcal{E}\right] &\le \sum_{t=1}^{T}\frac{2B_f}{t} + \sum_{t=1}^{T}(\frac{B_g^2}{V_t} + \frac{3kB_g}{\sqrt{tV_t}}) + \sum_{t=1}^{T}(\frac{kB_g}{V_t} + \frac{3k^2}{\sqrt{tV_t}}) + \sum_{t=1}^{T}(\frac{B_g^2+k^2}{tV_t}) \\ &\le 2B_f(\log T + 1) + \frac{16B_fB_g^2}{\delta}\sqrt{T} + \frac{24kB_fB_g}{\delta}(\log T + 1) + \frac{16kB_fB_g}{\delta}\sqrt{T} \\ &\quad + \frac{24k^2B_f}{\delta}(\log T + 1) + \frac{4B_f(B_g^2+k^2)}{\delta\sqrt{T}}\end{aligned}$$

where the first inequality holds since $\epsilon_t = \frac{k}{\sqrt{t}} \le k$. Therefore, we prove Lemma 6 with $p = 1/T^2$ that

$$\begin{aligned}\mathbb{E}[\sum_{t=1}^{T} \check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}] \le\ & \mathbb{E}[\sum_{t=1}^{T} \check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}|\mathcal{E}] + T\mathbb{P}(c \in C_p)B_G p \\ \le\ & (2B_f + \frac{24(kB_fB_g + k^2B_f)}{\delta})(1 + \log T) + \frac{16B_fB_g(B_g+1)}{\delta}\sqrt{T} \\ & ++\frac{4B_f(B_g^2+k^2)}{\delta\sqrt{T}} + \frac{\mathbb{P}(c \in C_p)B_G}{T}.\end{aligned}$$

E.2. Proving cumulative latency violation bound

Finally, we prove cumulative latency violation bound in Theorem 1.

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T} G^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}\right] \le\ & \mathbb{E}\left[\sum_{t=1}^{T} \check{G}_t^+(c_t, x_t)\mathbb{I}\{c_t \in C_p\}\right] + \mathbb{E}\left[\sum_{t=1}^{T}(G(c_t, x_t) - \check{G}_t(c_t, x_t))^+\mathbb{I}\{c_t \in C_p\}\right] \\ \le\ & (2B_f + \frac{24(kB_fB_g + k^2B_f)}{\delta})(1 + \log T) + \frac{16B_fB_g(B_g+1)}{\delta}\sqrt{T} \\ & ++\frac{4B_f(B_g^2+k^2)}{\delta\sqrt{T}} + \frac{\mathbb{P}(c \in C_p)B_G}{T}. + (2B_GTp + 2\beta_T^g\sqrt{4(T+2)\gamma_T^G}),\end{aligned}$$

where the last inequality holds since from Lemma 11

$$\mathbb{E}\left[\sum_{t=1}^{T} G(c_t, x_t) - \check{G}_t(c_t, x_t)\right] \le 2B_GTp + 2\beta_T^g\sqrt{4(T+2)\gamma_T^G}.$$

Appendix F. Additional details on the experiments

F.1. Modification for CKB algorithm in [14]

CKB in [14] also utilizes the Gaussian process to construct estimates for the utility function and constraint functions. However, it only considers to optimize the setting with single-type requests. We modify CKB such that it is applicable to the setting with multiple-type requests:

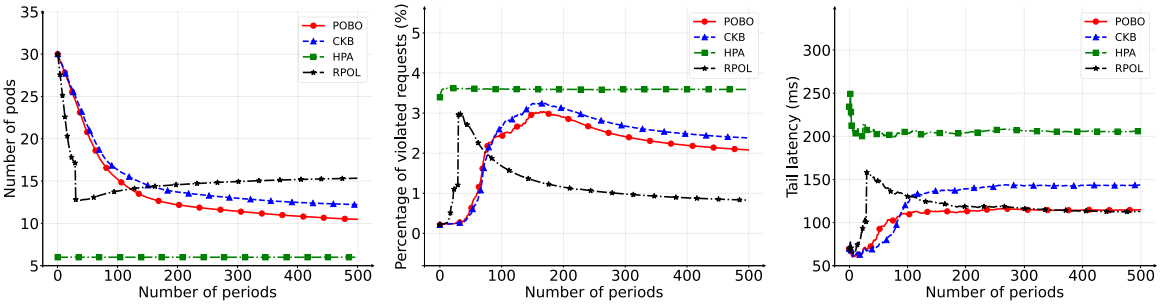
- Decision: When a c_t-type microservice request arrives, we use optimistic/pessimistic estimations for reward/constraints in the same way we define in Section 3. Then, we construct the pseudo-acquisition function and make the decision such that

  $$x_t = \arg\max_{x \in \mathcal{X}} \hat{f}_t(c, x) - \phi_t\check{g}_t(c, x) - \hat{\phi}_t(c)\check{G}_t(c, x),$$

  where $\hat{\phi}_t(c)$ is the dual variable corresponding to $G(c, x) \le 0$.
- Update: The dual variable $\hat{\phi}_{t+1}(c)$ updates similarly with ϕ_t as follows

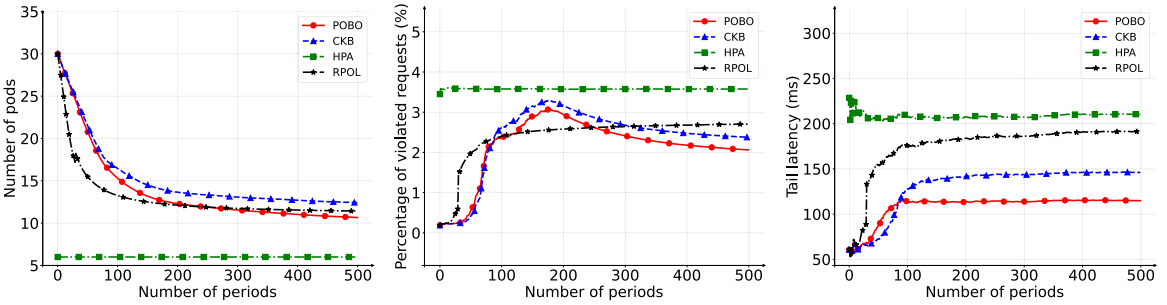
  $$\phi_{t+1}(c_t) = [\phi_{t+1}(c_t) + \bar{G}_t(c_t, x_t)/V]_0^\rho,$$

  where ρ is a hyperparameter related to the Slater's constant δ.



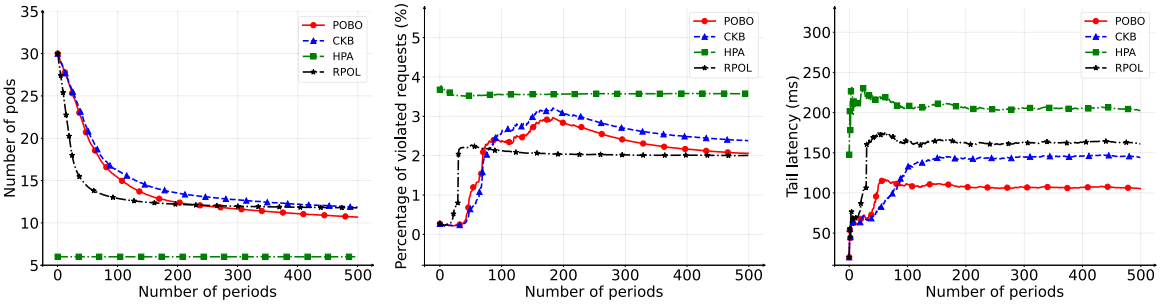
(a) Average number of used pods. (b) Percentage of violated requests. (c) P90 tail latency.

Fig. F.12. RPOL's target tail latency for all requests is set to 400 ms.



(a) Average number of used pods. (b) Percentage of violated requests. (c) P90 tail latency.

Fig. F.13. RPOL's target tail latency for all requests is set to 1000 ms.

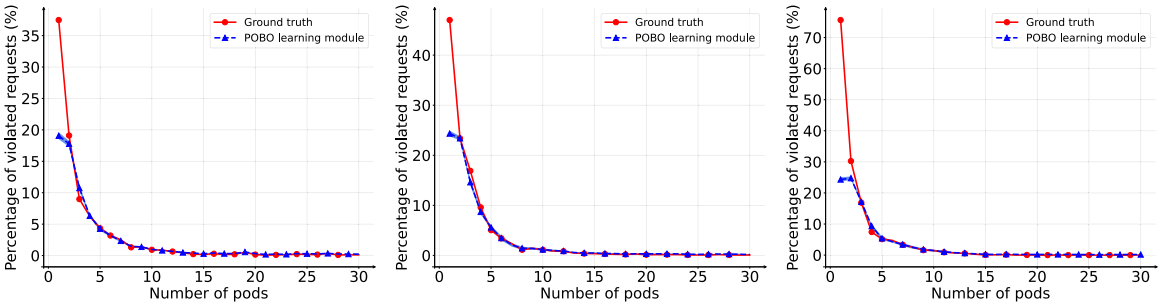


(a) Average number of used pods. (b) Percentage of violated requests. (c) P90 tail latency.

Fig. F.14. RPOL's target tail latencies for login/search/recommendation requests are set to 400/700/1000 ms.

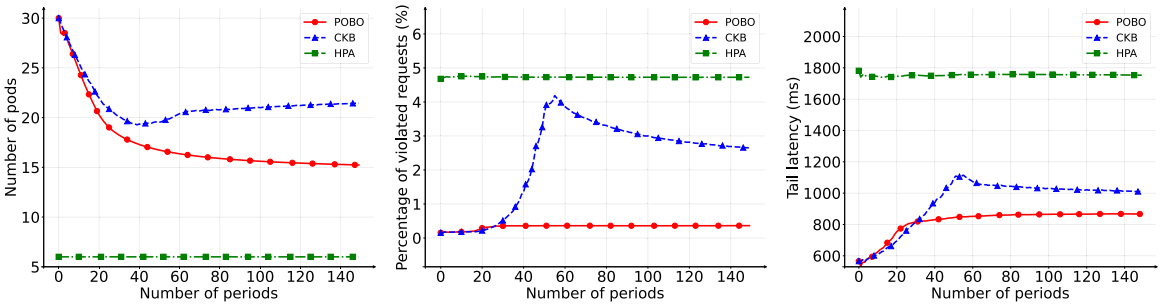
F.2. Discussion on RPOL

As discussed in the paper, RPOL is suitable for SLA on the tail latency. There are two issues in applying RPOL in our setting: (1) it is unclear how to properly set the threshold of the tail latency to satisfy the average system latency constraint; (2) RPOL might behave aggressively (or over-conservatively in satisfying SLAs). Given these two issues, we conduct preliminary experiments for interested readers where we customize RPOL for our setting by imposing various tail latencies to test the gap between RPOL and POBO. The setup is the same as in the experiments of stationary requests in Section 5.3.2. The results are summarized in Figs. F.12–F.14. For three settings, compared with RPOL, POBO has the average pod usage with gaps of −4.86, −0.76, and −1.10; percentage of violated requests with the gaps −1.25%, −0.63%, and 0.05%; and the P90 latency with the gaps −2.05, −76.22, and −55.82 ms. The results justify our intuition that a tight threshold leads to a low percentage of violated requests but might introduce the conservative resource configuration in Fig. F.12, and a loose threshold leads to a proper configuration but results in a large tail latency and a larger percentage of violated requests in Fig. F.13. The observation is also similar when we set different thresholds for three types of requests in Fig. F.14.



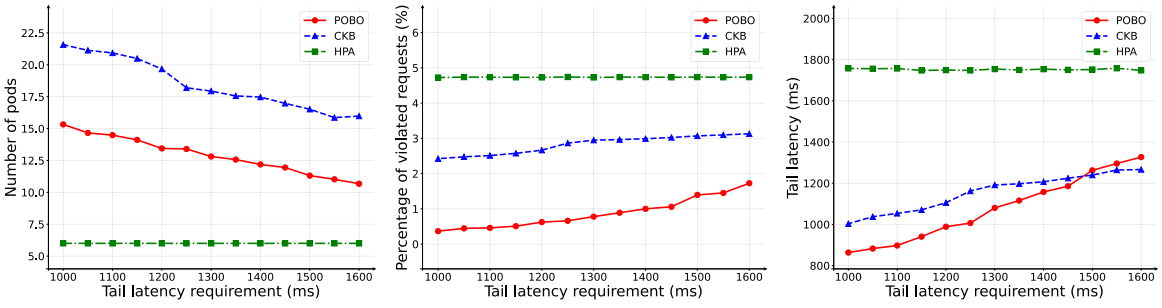
(a) Login (b) Recommendation (c) Search

Fig. F.15. Percentage of violated requests-resource function learning.



(a) average number of used pods. (b) percentage of violated requests. (c) P90 tail latency of requests.

Fig. F.16. Warm-up test with single-type requests: search requests.



(a) average number of used pods. (b) percentage of violated requests. (c) P90 tail latency.

Fig. F.17. Sensitivity study with just search requests.

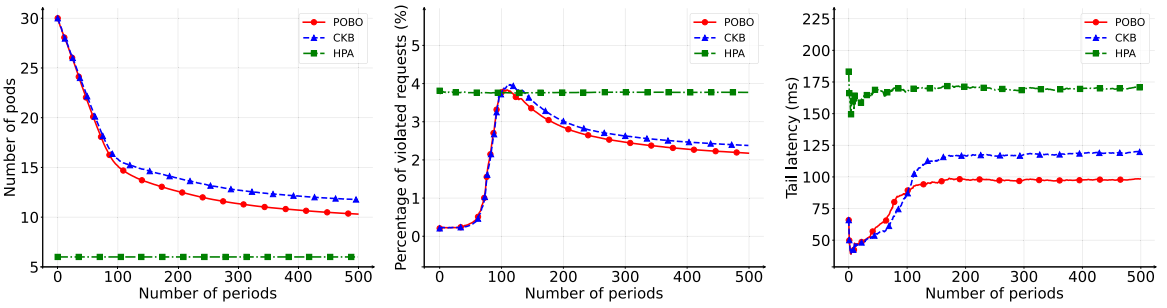
F.3. Learning curve for the percentage of violated requests constraint functions

We also test the ability of POBO to learn the relationship between the percentage of violated requests and the number of pods. Fig. F.15 shows that the function learned by POBO is very close to the ground truth.

F.4. The experiment with just single-type requests: Search requests

We test the POBO with just search requests. Fig. F.16 plots the performance under HPA, CKB, and POBO for the search requests, respectively. Compared with CKB, POBO has a lower average pod usage with a gap of 6.20; a lower percentage of violated requests with a gap of 2.28%, and a better P90 latency with a gap of 143.26 ms.

We also perform a sensitivity study with the search requests. Fig. F.17 plots the performance for the search requests with the tail latency requirements from the strict 1000 ms to the loose 1600 ms. Since the complexity and average latency of search requests is high, the requirement range of search requests is larger than login and recommendation. Results verify that POBO is robust and adaptive to the varying SLA requirement on tail latency with the search requests.



(a) average number of used pods. (b) percentage of violated requests. (c) P90 tail latency.

Fig. F.18. The experiment under stationary distribution of [33%, 33%, 33%].

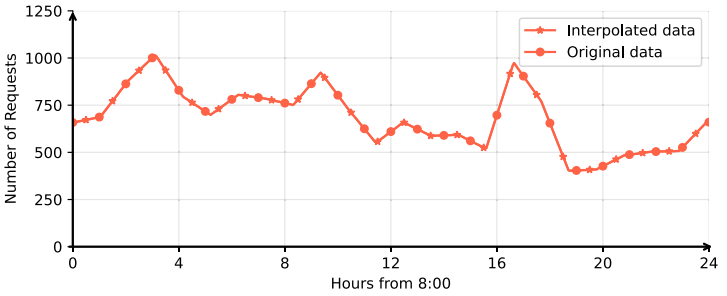


Fig. F.19. The daily workload pattern extracted and interpolated from [27].

F.5. The experiment for the multi-type requests under the distribution [33%, 33%, 33%]

We consider the requests for login, recommendation, and search generated from the distribution of [33%, 33%, 33%], which is a uniform distribution. Figs. F.18(a)–F.18(c) plot the performance under HPA, CKB, and POBO for the workload. Compared with CKB, POBO has a lower average pod usage with a gap of 1.46; a lower percentage of violated requests with a gap of 0.20% and a better P90 latency with a gap of 21.44 ms.

F.6. Non-stationary traffic pattern in [27]

As in [27], we analyze the traffic pattern of a workload trace collected from a production MLaaS cluster in Alibaba. We extract the number of tasks submitted for the day after 8:00 a.m. Since the raw data was recorded hourly and had only 24 values, we linearly interpolate the raw data by plugging in 20 values every hour. The trajectory is shown in Fig. F.19.

References

[1] N. Alshuqayran, N. Ali, R. Evans, A systematic mapping study in microservice architecture, in: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA, IEEE, 2016, pp. 44–51.
[2] Alibaba cloud, 2023, URL https://www.alibabacloud.com.
[3] Amazon web services, 2023, URL https://aws.amazon.com.
[4] GoogleCloud, 2023, URL https://cloud.google.com.
[5] J. Park, B. Choi, C. Lee, D. Han, GRAF: A graph neural network based proactive resource allocation framework for SLO-oriented microservices, in: Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies, 2021, pp. 154–167.
[6] Y. Zhang, W. Hua, Z. Zhou, G.E. Suh, C. Delimitrou, Sinan: ML-based and qos-aware resource management for cloud microservices, in: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2021, pp. 167–181.
[7] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, J. He, G. Yang, C. Xu, Erms: Efficient resource management for shared microservices with SLA guarantees, in: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, 2022, pp. 62–77.
[8] Q. Li, B. Li, P. Mercati, R. Illikkal, C. Tai, M. Kishinevsky, C. Kozyrakis, RAMBO: Resource allocation for microservices using Bayesian optimization, IEEE Comput. Archit. Lett. 20 (1) (2021) 46–49.
[9] T. Patel, 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.
[10] R.B. Roy, T. Patel, D. Tiwari, Satori: efficient and fair resource partitioning by sacrificing short-term benefits for long-term gains, in: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2021, pp. 292–305.
[11] Y. Liu, H. Xu, W.C. Lau, Online resource optimization for elastic stream processing with regret guarantee, in: Proceedings of the 51st International Conference on Parallel Processing, 2022, pp. 1–11.

[12] Z. Zhou, Y. Zhang, C. Delimitrou, AQUATOPE: QoS-and-uncertainty-aware resource management for multi-stage serverless workflows, in: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, 2022, pp. 1–14.
[13] X. Zhang, H. Wu, Y. Li, J. Tan, F. Li, B. Cui, Towards dynamic and safe configuration tuning for cloud databases, in: Proceedings of the 2022 International Conference on Management of Data, 2022, pp. 631–645.
[14] X. Zhou, B. Ji, On kernelized multi-armed bandits with constraints, in: Thirty-Sixth Conference on Neural Information Processing Systems, 2022.
[15] H. Guo, Q. Zhu, X. Liu, Rectified pessimistic-optimistic learning for stochastic continuum-armed bandit with constraints, 2022, arXiv preprint arXiv: 2211.14720.
[16] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, et al., An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 3–18.
[17] S.R. Chowdhury, A. Gopalan, On kernelized multi-armed bandits, in: International Conference on Machine Learning, PMLR, 2017, pp. 844–853.
[18] Jaeger, 2023, URL https://www.jaegertracing.io/.
[19] X. Liu, B. Li, P. Shi, L. Ying, An efficient pessimistic-optimistic algorithm for stochastic linear bandits with general constraints, Adv. Neural Inf. Process. Syst. 34 (2021) 24075–24086.
[20] M.J. Neely, Energy-aware wireless scheduling with near-optimal backlog and convergence time tradeoffs, IEEE/ACM Trans. Netw. 24 (4) (2015) 2223–2236.
[21] Docker desktop, 2023, URL https://www.docker.com/products/docker-desktop/.
[22] Kubernetes, 2023, URL https://kubernetes.io/.
[23] Kubernetes python client, 2023, URL https://github.com/kubernetes-client/python.
[24] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, C. Delimitrou, An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, 2019, pp. 3–18.
[25] Wrk2, 2023, URL https://github.com/giltene/wrk2.
[26] Kubernetes horizontal pod autoscaling, 2023, URL https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/.
[27] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, Y. Ding, Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters, in: 19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022, 2022, pp. 945–960.
[28] N. Srinivas, A. Krause, S.M. Kakade, M. Seeger, Gaussian process optimization in the bandit setting: No regret and experimental design, in: Proceedings of the International Conference on Machine Learning, 2010, 2010.
[29] B. Hajek, Hitting-time and occupation-time bounds implied by drift analysis with applications, Adv. Appl. Probab. 14 (3) (1982) 502–525.