Joint Pricing and Scheduling for SLA-aware Serverless GPU Services

Xiaoyao Huang*, Yanan Yang*, Yujun Wang§, and Jie Wu*‡
*Cloud Computing Research Institute, China Telecom, China
‡Department of Computer and Information Sciences, Temple University, USA
§China Telecom Cloud Technology Co. Ltd, China
Email: huangxy32@chinatelecom.cn, yangyn11@chinatelecom.cn, jiewu@temple.edu, wangyj3@chinatelecom.cn

***Abstract*—The growing demand for large-scale AI inference drives the need for elastic and profitable GPU-as-a-Service platforms. However, existing methods often decouple pricing and scheduling, resulting in reactive and inefficient resource utilization. This paper presents HOPS, a hierarchical optimization framework that jointly integrates dynamic pricing and real-time scheduling for SLA-aware serverless GPU services. The upper-level Pricing and Baseline Adjustment (PBA) algorithm adaptively learns per-tier prices and baseline GPU allocations via stochastic gradient updates, while the lower-level Adaptive Reconfiguration and Matching Scheduler (ARMS) manages fine-grained job placement to minimize SLA violations and switching overhead. The two layers form a closed feedback loop to achieve equilibrium between economic and operational objectives. Trace-driven experiments using real workloads show that HOPS improves overall profit by 45–55%, reduces SLA violations by up to 30%, and lowers switching overhead by one order of magnitude, while sustaining over 70% GPU utilization.**

***Index Terms*—Serverless pricing, GPU scheduling, Stackelberg game, bi-level optimization.**

I. Introduction

The rapid growth of AI-generated content (AIGC) and large language model (LLM) inference has driven a surge in serverless GPU services, which provide elastic, pay-as-you-go GPU access for workloads such as LLM inference, multimodal generation, and real-time analytics [1], [2]. However, demand is highly bursty and time-varying [3], while GPUs are expensive and scarce, making even short periods of under-utilization costly [4]. Meanwhile, stringent service-level agreements (SLAs) impose latency and reliability requirements, where violations can incur substantial penalties. These factors make it challenging to jointly balance profitability, utilization, and SLA compliance in serverless GPU platforms.

Most existing work focuses on low-level scheduling given fixed demand. For example, MQFQ-Sticky improves fairness and mitigates cold starts through multi-queue scheduling and GPU memory management [5], while HAS-GPU enables fine-grained GPU slicing to meet SLA targets under heterogeneous workloads [6]. While effective in improving utilization and responsiveness, these methods treat demand as exogenous and decouple pricing from scheduling, leaving providers largely reactive to workload surges. As a result, platforms may over-provision and waste GPUs, or under-provision and suffer SLA violations and revenue loss.

To address this limitation, we propose HOPS, a joint Pricing $\times$ Scheduling framework that integrates dynamic pricing with real-time scheduling. Pricing steers tier choices and reshapes workload distribution, while co-optimized scheduling executes requests efficiently under SLA constraints. By coupling the two layers, HOPS enables the platform to proactively balance profitability, resource utilization, and service quality under heterogeneous and rapidly changing workloads.

The main contributions are threefold. First, we formulate the pricing–scheduling interaction as a bi-level optimization problem, where pricing is optimized at a longer planning scale and scheduling operates at a fine-grained real-time scale. Second, we design HOPS, a hierarchical algorithm that combines stochastic optimization for pricing with adaptive reconfiguration and matching for scalable scheduling. Finally, trace-driven experiments on real-world traces show that HOPS improves GPU utilization and profit while reducing SLA violations and switching overhead compared with state-of-the-art baselines such as MQFQ-Sticky and HAS-GPU [5], [6].

II. System Model and Problem Formulation

A. System Overview

As illustrated in Fig. 1, we consider a serverless GPU platform that serves a set of users $\mathcal{U}$. The system operates on two coupled timescales: a *window level* for long-term economic planning and a *slot level* for real-time resource control. Time is divided into windows indexed by t, each containing slots indexed by $h \in \mathcal{H}_t$.

At the beginning of each window, the *Pricing Layer* announces a per-unit virtual GPU (vGPU) price $p_{m,t}$ and an SLA guarantee s_m for each service tier $m \in \mathcal{M} = \{1, 2, \ldots, M\}$. Users then select a tier and submit their workloads. Each tier is initialized with a minimum number of persistent *keep-alive replicas* to ensure zero cold-start latency and baseline availability. During the window, subscribed users generate jobs following a stochastic process; for tier m, the aggregate arrival rate is denoted by $\lambda_{m,t}$ and the *average vGPU demand per job* by $g_{m,t}$, yielding expected demand $\lambda_{m,t} g_{m,t}$.

At the slot level, the *Scheduling Layer* assigns incoming jobs to vGPUs and may reconfigure which function each vGPU runs. Such reconfiguration causes switching overhead, so the scheduler balances SLA compliance against switching and usage costs. The vGPU pool is logically divided into

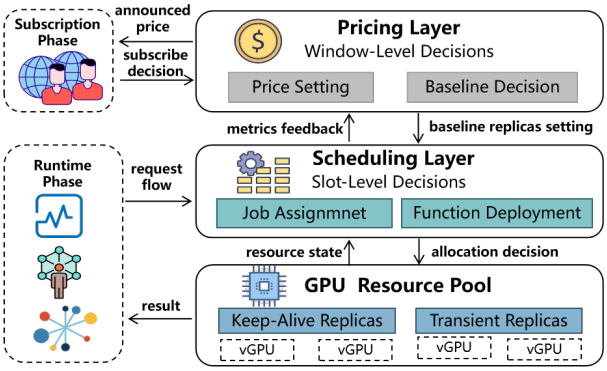


Fig. 1: The system framework.

persistent keep-alive replicas, which remain online throughout the window, and elastic *transient replicas*, which can be activated temporarily to handle bursty workloads and released as demand subsides. At each window end, metrics (SLA violations, utilization) are aggregated and fed back to the Pricing Layer to update $p_{m,t+1}$ and baseline configurations, forming a closed loop under dynamic, heterogeneous workloads.

B. Pricing Layer Modeling

At the start of window t, the Pricing Layer determines the per-unit vGPU price $p_{m,t}$ and the number of keep-alive replicas $n_{m,t}^{\min}$ for each tier m. The SLA guarantee s_m specifies the maximum job completion time in tier m. The baseline $n_{m,t}^{\min}$ provides always-on capacity to eliminate startup delays.

Let Λ_t be the total potential arrival rate during window t. The realized rate $\lambda_{m,t}$ depends on the announced prices and SLAs. We adopt a multinomial logit (MNL) model [7]:

$$\lambda_{m,t} = \Lambda_t \cdot \frac{\exp(\alpha s_m - \beta p_{m,t})}{\sum_{m' \in \mathcal{M}} \exp(\alpha s_{m'} - \beta p_{m',t})}, \tag{1}$$

where α and β capture user sensitivity to SLA and price, respectively. The expected total demand of tier m over the window is $\lambda_{m,t} g_{m,t} \Delta t$.

The provider's revenue is $R_t = \sum_{m \in \mathcal{M}} p_{m,t} \lambda_{m,t} g_{m,t} \Delta t$, and the total operational cost is

$$C_t = C_t^{\text{base}} + C_t^{\text{gpu}} + C_t^{\text{switch}} + \sum\nolimits_{m \in \mathcal{M}} b_m L_{m,t}, \tag{2}$$

where $C_t^{\text{base}} = \sum_m r_m n_{m,t}^{\min}$ is the cost of keep-alive replicas with per-replica baseline cost of a tier r_m, C_t^{gpu} is the transient vGPU usage cost, C_t^{switch} denotes the overhead when *keep-alive vGPUs* change their deployed functions across slots, and $L_{m,t}$ counts SLA-violated jobs with per-job penalty b_m.

The Pricing Layer maximizes the cumulative profit across all windows

$$\max_{\mathbf{p}_t,\, \mathbf{n}_t^{\min}} \quad \Phi = \sum\nolimits_t \big(R_t - C_t\big), \tag{3}$$

subject to the capacity constraint

$$\text{s.t.} \quad \sum\nolimits_{m \in \mathcal{M}} n_{m,t}^{\min} \le N_t, \quad \forall t, \tag{4}$$

where N_t is the total number of available vGPUs in window t. The quantities C_t^{gpu}, C_t^{switch}, and $L_{m,t}$ are induced by slot-level scheduling decisions described next.

C. Scheduling Layer Modeling

The Scheduling Layer operates at the slot level. Let $\mathcal{J}$ be the set of vGPUs and $\mathcal{F}$ the set of functions. Each arriving job k in slot h requires function $f(k) \in \mathcal{F}$ and belongs to tier $m(k) \in \mathcal{M}$. We use $x_{k,j,h}$ for *Job Assignment* (job k assigned to vGPU j in slot h), $v_{j,f,h}$ for *Function Deployment* (vGPU j runs function f in slot h), and $z_{j,h}$ to indicate a function switch on vGPU j between slots $h-1$ and h.

Given $\mathbf{p}_t$ and $\mathbf{n}_t^{\min}$, the Scheduling Layer minimizes

$$\min_{x,v,z} \quad C_t^{\text{gpu}} + C_t^{\text{switch}} + \sum\nolimits_{m \in \mathcal{M}} b_m L_{m,t}, \tag{5}$$

with $C_t^{\text{gpu}} = c^{\text{gpu}} \sum_{h \in \mathcal{H}_t} \sum_{j \in \mathcal{J}} \sum_{f \in \mathcal{F}} v_{j,f,h}$ and $C_t^{\text{switch}} = c^{\text{switch}} \sum_{h \in \mathcal{H}_t} \sum_{j \in \mathcal{J}} z_{j,h}$. Feasibility requires that a job can be served only by a vGPU running its required function,

$$x_{k,j,h} \le v_{j,f(k),h}, \quad \forall k, j, h, \tag{6}$$

each vGPU hosts at most one function per slot,

$$\sum\nolimits_{f \in \mathcal{F}} v_{j,f,h} \le 1, \quad \forall j, h, \tag{7}$$

switching is captured by

$$\sum\nolimits_{f \in \mathcal{F}} |v_{j,f,h} - v_{j,f,h-1}| \le 2\, z_{j,h}, \quad \forall j, h > h_0, \tag{8}$$

each vGPU processes at most one job per slot,

$$\sum\nolimits_{k} x_{k,j,h} \le 1, \quad \forall j, h, \tag{9}$$

and the number of active vGPUs mapped to functions of tier m never falls below the baseline,

$$\sum\nolimits_{j \in \mathcal{J}} \sum\nolimits_{f:\, m(f)=m} v_{j,f,h} \ge n_{m,t}^{\min}, \quad \forall m, h. \tag{10}$$

Each job k has completion time T_k subject to $T_k \le s_{m(k)} + \delta_k$, where $\delta_k \ge 0$ is a slack variable. The number of SLA-violated jobs in tier m is $L_{m,t} = \sum_{k:m(k)=m} \mathbb{I}[\delta_k > 0]$. At the beginning of each window, $\mathbf{n}_t^{\min}$ initializes the persistent pool; within the window, the scheduler adjusts $v_{j,f,h}$ by activating transient replicas or switching functions on keep-alive replicas. The realized C_t^{gpu}, C_t^{switch}, and $L_{m,t}$ are reported to the Pricing Layer for the next window.

D. Stackelberg Formulation

The two layers form a Stackelberg game. The Pricing Layer acts as the leader, deciding $\mathbf{p}_t$ and $\mathbf{n}_t^{\min}$ at each window start while anticipating the Scheduling Layer's response. The Scheduling Layer is the follower, solving (5)–(10) given the upper-level decisions. Let $\Psi(\mathbf{p}_t, \mathbf{n}_t^{\min})$ denote the optimal objective value of the Scheduling Layer. The problem is

$$\max_{\mathbf{p}_t, \mathbf{n}_t^{\min}} \quad \Phi = \sum_t \big(R_t - \Psi(\mathbf{p}_t, \mathbf{n}_t^{\min})\big), \tag{11}$$

$$\text{s.t.} \quad \sum\nolimits_{m \in \mathcal{M}} n_{m,t}^{\min} \le N_t, \quad \forall t.$$

This hierarchical structure captures demand shaping at the upper level and real-time resource control at the lower level, yielding a joint pricing-and-scheduling strategy that maximizes profit while ensuring SLA compliance.

III. Proposed Algorithms

In this section, we propose a hierarchical framework named *Hierarchical Optimization for Pricing and Scheduling (HOPS)* to jointly optimize the upper-level Pricing Layer and the lower-level Scheduling Layer. The two layers operate at different timescales: the Pricing Layer determines per-tier prices and baseline vGPU allocations at a slower *window level*, while the Scheduling Layer dynamically manages function deployment and job allocation at a faster *slot level*. Directly solving the integrated problem is intractable because the Scheduling Layer involves combinatorial integer decisions and the Pricing Layer contains continuous variables. To address this, HOPS decomposes the problem into two nested loops: the lower-level Scheduling Layer acts as a follower that computes the optimal or approximate response to given prices and baselines, while the upper-level Pricing Layer updates its strategy based on aggregated feedback. This structure naturally aligns with a Stackelberg game, where the leader iteratively adjusts its strategy and the follower responds, enabling scalability and convergence to equilibrium.

A. Window-Level Algorithm: Pricing and Baseline Adjustment (PBA)

At the beginning of each window, the Pricing Layer determines the per-tier price vector $\mathbf{p}_t = \{p_{m,t}\}$ and the baseline configuration $\mathbf{n}_t^{\min} = \{n_{m,t}^{\min}\}$, based on historical demand and the realized costs returned by the Scheduling Layer in the previous window. The goal is to maximize the profit function Φ_t defined in (3). However, the lower-level Scheduling Layer problem is discrete and highly non-linear, making it impossible to obtain a closed-form gradient of Φ_t with respect to $(\mathbf{p}_t, \mathbf{n}_t^{\min})$.

To overcome this challenge, we adopt a *blockwise two-point stochastic gradient estimation* scheme combined with a projected gradient update. Let $\Delta_t = (\Delta_t^p, \Delta_t^n)$ be a random perturbation vector, where each entry independently takes values in $\{\pm 1\}$ (Rademacher perturbation). Given a perturbation magnitude $c > 0$, two perturbed strategies are generated:

$$\mathbf{s}^+ = (\mathbf{p}_t + c\Delta_t^p,\ \mathbf{n}_t^{\min} + c\Delta_t^n), \quad \mathbf{s}^- = (\mathbf{p}_t - c\Delta_t^p,\ \mathbf{n}_t^{\min} - c\Delta_t^n).$$

The Scheduling Layer evaluates the profit at these two points, denoted by Φ^+ and Φ^-, and the gradient is approximated as

$$\widehat{\nabla}\Phi_t = \Delta_t(\Phi^+ - \Phi^-)/2c, \tag{12}$$

where the multiplication is element-wise.

The estimate in (12) is then used to update the decision variables via a projected gradient step:

$$(\mathbf{p}_{t+1}, \mathbf{n}_{t+1}^{\min}) = \Pi\big[(\mathbf{p}_t, \mathbf{n}_t^{\min}) + \eta_t \widehat{\nabla}\Phi_t\big], \tag{13}$$

where η_t is the step size and $\Pi(\cdot)$ is the projection operator ensuring feasibility: $\sum_m n_{m,t}^{\min} \le N_t, \quad n_{m,t}^{\min} \in \mathbb{Z}_{\ge 0}, \quad p_{m,t} \ge 0$. We implement Π blockwise: prices are projected by non-negativity and optional upper bounds, while baselines are rounded to the nearest integers and then greedily adjusted to satisfy the total vGPU capacity constraint $\sum_m n_{m,t}^{\min} \le N_t$[1].

Algorithm 1: Pricing and Baseline Adjustment (PBA)

Input: Initial $\mathbf{p}_0$, $\mathbf{n}_0^{\min}$, step sizes $\{\eta_t\}$, perturbation magnitude c, total horizon T.

for $t = 1, 2, \ldots, T$ **do**
- Generate random perturbation vector Δ_t with Rademacher entries;
- Compute perturbed strategies: $\mathbf{s}^+ = (\mathbf{p}_t + c\Delta_t^p,\ \mathbf{n}_t^{\min} + c\Delta_t^n)$, $\mathbf{s}^- = (\mathbf{p}_t - c\Delta_t^p,\ \mathbf{n}_t^{\min} - c\Delta_t^n)$;
- Evaluate profits Φ^+ and Φ^- by invoking Scheduling Algorithm (ARMS) under $\mathbf{s}^+$ and $\mathbf{s}^-$;
- Approximate gradient $\widehat{\nabla}\Phi_t$ using (12);
- Update variables using projected step (13);
- Pass $(\mathbf{p}_{t+1}, \mathbf{n}_{t+1}^{\min})$ to the Scheduling Layer for the next window.

The PBA algorithm is summarized in Algorithm 1. The updated prices and baseline decisions are passed to the Scheduling Layer at the start of the next window, forming a closed feedback loop between the two layers.

B. Slot-Level Algorithm: Adaptive Reconfiguration and Matching Scheduler (ARMS)

We design ARMS to solve the lower-level scheduling problem by dynamically determining $v_{j,f,h}$, $z_{j,h}$, and $x_{k,j,h}$ in each slot h of window t, minimizing

$$C_t^{\text{gpu}} + C_t^{\text{switch}} + \sum\nolimits_m b_m L_{m,t},$$

subject to (6)–(10). The MILP is intractable for large-scale systems, so ARMS adopts a two-stage approach: (i) *Stage 1: Reconfiguration* to adjust active vGPUs, and (ii) *Stage 2: Matching* to assign jobs.

Stage 1: Fairness-aware reconfiguration. At the start of slot h, the required replicas for each function f are estimated as

$$n_{f,h}^{\text{req}} = \left\lceil (Q_{f,h} + \hat{\lambda}_{f,h}\,\Delta_h)/\mu_f\,\Delta_h \right\rceil, \tag{14}$$

where $Q_{f,h}$ is the queue length of function f, μ_f is the service rate per vGPU. A fairness score

$$\text{score}_f = (n_{f,h}^{\text{req}} - n_{f,h}^{\text{active}})_+ + \gamma D_{f,h},$$

with $n_{f,h}^{\text{active}} = \sum_j v_{j,f,h}$ and deficit $D_{f,h}$, prioritizes underserved functions. When scaling up, ARMS selects between switching a keep-alive vGPU or activating a transient one:

$$\Delta C_{\text{switch}}(j,h) = c^{\text{switch}} + \gamma Q_{j,h}^{\text{remain}}, \quad \Delta C_{\text{act}} = c^{\text{gpu}},$$

Make the decision to choose the cheaper action while preserving baseline constraints.

Stage 2: Value- and fairness-aware matching. Given $v_{j,f,h}$, jobs are scheduled by priority

$$\begin{aligned}\pi_k =& (p_{m(k),t}\mathbb{E}[g_{m(k),t}] + \kappa b_{m(k)}) \\ & \cdot (1/(\max\{s_{m(k)} - t_h, \epsilon\})) \cdot (1 + \eta \widetilde{D}_{f(k),h}),\end{aligned} \tag{15}$$

[1]When generating $\mathbf{s}^{\pm}$, any infeasible perturbed baselines are projected onto the feasible integer simplex before evaluation, ensuring both perturbed points can be processed by the Scheduling Layer. This keeps the two-point estimator in (12) valid without altering the algorithm structure.

Algorithm 2: ARMS: Adaptive Reconfiguration and Matching Scheduler

Input: $\mathbf{p}_t$, $\mathbf{n}_t^{\min}$, $\{\lambda_{m,t}\}$, $\mathcal{H}_t$, $\mathcal{J}$, $\mathcal{F}$.
Output: C_t^{gpu}, C_t^{switch}, $\{L_{m,t}\}$.

Initialize v_{j,f,h_0} so that $\sum_j \sum_{f:m(f)=m} v_{j,f,h_0} \geq n_{m,t}^{\min}$.
for $h \in \mathcal{H}_t$ **do**
  Update $Q_{f,h}$ and $\hat{\lambda}_{f,h}$.
  for $f \in \mathcal{F}$ *in descending* score_f **do**
    while $\sum_j v_{j,f,h} < n_{f,h}^{req}$ **do**
      Identify baseline-safe keep-alive set $\mathcal{J}_{f,h}^{\text{sw}}$.
      if $\mathcal{J}_{f,h}^{sw} \neq \emptyset$ *and* $\min_j \Delta C_{switch}(j,h) < \Delta C_{act}$ **then**
        Switch $j^\star = \arg\min_j \Delta C_{\text{switch}}(j,h)$.
      else
        Activate a transient vGPU $j^\dagger$.
  Build job set $\mathcal{K}_h$, sort by π_k.
  for *job* k *in sorted order* **do**
    Assign to $j^\star = \arg\min_j \widehat{T}_{k,j,h}$ if feasible; else mark violation or queue.
  Update C_t^{gpu}, C_t^{switch}.

where the three factors respectively represent economic value, SLA urgency, and fairness. Ties are resolved by minimum predicted completion time

$$\widehat{T}_{k,j,h} = t_h + (q_{j,h}+1)/\mu_{f(k)}, \tag{16}$$

where $q_{j,h}$ is the queue length on vGPU j. Jobs with $\widehat{T}_{k,j,h} > s_{m(k)}$ are marked as SLA violations.

Stage 1 executes in $O(|\mathcal{F}||\mathcal{J}|)$ time per slot, while Stage 2 operates in $O(|\mathcal{K}_h|\log|\mathcal{K}_h|)$. Given fixed function placements $v_{j,f,h}$, Stage 2 achieves optimal scheduling under the defined priority scheme, whereas Stage 1 yields a bounded approximation. Together, they guarantee overall efficiency, fairness, and SLA compliance across dynamic workloads.

C. Integrated Bi-Level Optimization Framework

The proposed HOPS framework tightly couples the window-level PBA algorithm with the slot-level ARMS to jointly optimize long-term economic objectives and real-time resource control. This naturally yields a bi-level leader–follower structure: at the beginning of each window, the upper-level leader (PBA) sets the per-tier price vector $\mathbf{p}_t = \{p_{m,t}\}$ and baseline keep-alive vGPU allocation $\mathbf{n}_t^{\min} = \{n_{m,t}^{\min}\}$, while during the window, the lower-level follower (ARMS) dynamically manages function deployment and job scheduling.

Upper level. Given the realized costs from the previous window, PBA maximizes the cumulative profit:

$$\max_{\mathbf{p}_t,\, \mathbf{n}_t^{\min}} \quad \Phi = \sum\nolimits_t \big(R_t - C_t\big), \tag{17}$$

subject to GPU capacity and feasibility:

$$\sum\nolimits_{m\in\mathcal{M}} n_{m,t}^{\min} \leq N_t, \qquad n_{m,t}^{\min} \in \mathbb{Z}_{\geq 0}, \qquad p_{m,t} \geq 0,$$

where R_t and C_t are defined in Section III-A and C_t depends on ARMS decisions in window t, coupling the two levels.

Lower level. Given $(\mathbf{p}_t, \mathbf{n}_t^{\min})$, ARMS minimizes the real-time operational cost:

$$\begin{aligned} \min_{x,v,z} \quad & C_t^{\text{gpu}} + C_t^{\text{switch}} + \sum\nolimits_{m\in\mathcal{M}} b_m L_{m,t}, \\ \text{s.t.} \quad & (6),\ (7),\ (8),\ (9),\ (10), \end{aligned} \tag{18}$$

where $x_{k,j,h}$, $v_{j,f,h}$, and $z_{j,h}$ are slot-level variables defined in Section III-B. The follower determines job allocation, transient vGPU activation, and switching of keep-alive vGPUs while respecting baseline guarantees.

Closed-loop interaction. At the start of window t, PBA selects $(\mathbf{p}_t, \mathbf{n}_t^{\min})$. ARMS then operates over all slots $h \in \mathcal{H}_t$, generating realized metrics $\{C_t^{\text{gpu}}, C_t^{\text{switch}}, L_{m,t}\}$ and feeding them back to PBA, which updates its strategy using the projected stochastic update (13). Through repeated iterations, the leader adapts prices and baseline allocations to the follower's approximate best responses, driving the coupled system toward a Stackelberg-stable operating point.

D. Theoretical Analysis

We analyze the theoretical properties of HOPS, focusing on convergence and equilibrium stability.

Definition 1 (Stackelberg Equilibrium (SE))**.** *A strategy tuple* $(\mathbf{p}^\star, \mathbf{n}^{\min,\star}, x^\star, v^\star, z^\star)$ *constitutes a* SE *if:*

1) ***(Follower optimality)*** *For fixed* $(\mathbf{p}^\star, \mathbf{n}^{\min,\star})$, $(x^\star, v^\star, z^\star)$ *solves the follower's problem in* (18)*;*
2) ***(Leader optimality)*** *Given the follower's best-response mapping* $\mathcal{R}(\mathbf{p}, \mathbf{n}^{\min})$, $(\mathbf{p}^\star, \mathbf{n}^{\min,\star})$ *solves the leader's problem in* (17)*.*

Theorem 1 (Convergence of PBA)**.** *If the step sizes* $\{\eta_t\}$ *satisfy* $\sum_t \eta_t = \infty$, $\sum_t \eta_t^2 < \infty$, *and the perturbation magnitude* c *is bounded, then the iterates* $(\mathbf{p}_t, \mathbf{n}_t^{\min})$ *generated by the projected two-point update* $(\mathbf{p}_{t+1}, \mathbf{n}_{t+1}^{\min}) = \Pi\Big[(\mathbf{p}_t, \mathbf{n}_t^{\min}) + \eta_t \widehat{\nabla}\Phi_t\Big]$ *converge almost surely to a stationary point of the upper-level objective* Φ.

Proof. The estimator in (12) provides an unbiased stochastic approximation of $\nabla\Phi_t$. Under bounded variance and Lipschitz smoothness, the projected updates define a Robbins–Monro process that converges a.s. to the stationary set of Φ. □

Let $\mathcal{C}_{\text{ARMS}}$ and $\mathcal{C}_{\text{OPT}}$ denote the achieved and optimal objectives of the lower-level problem (18), respectively.

Theorem 2 (Approximation Guarantee of ARMS)**.** *For a fixed leader strategy* $(\mathbf{p}_t, \mathbf{n}_t^{\min})$*, the ARMS produces a solution* (x_t, v_t, z_t) *satisfying* $\mathcal{C}_{\text{ARMS}} \leq (1+\epsilon)\mathcal{C}_{\text{OPT}}$, *where* $\epsilon \in [0,1)$ *depends on workload heterogeneity and vGPU contention.*

Proof. ARMS decouples the follower decision into two stages: Stage 1 determines a feasible deployment by greedily reconfiguring vGPU allocations while explicitly trading off activation versus switching, which bounds the reconfiguration overhead relative to the minimum changes needed to meet the baseline constraints. Conditioned on this deployment, Stage 2 reduces to within-window priority-based job matching under capacity constraints; the induced objective is monotone

with diminishing returns, and the resulting greedy/prioritized matching achieves a $(1+\epsilon)$-approximation for the scheduling subproblem. Combining the two stages yields the stated bound, where ϵ captures the residual loss due to heterogeneous service rates and contention across tiers/vGPUs. □

Theorem 3 (Bi-Level Convergence)**.** *Consider the joint system state* $\Theta_t = (\mathbf{p}_t, \mathbf{n}_t^{\min}, x_t, \upsilon_t, z_t)$. *If the upper-level PBA operates on a slower timescale and ARMS produces bounded-error responses, then* $\lim_{t\to\infty} \mathrm{dist}(\Theta_t, \Theta^\star) = 0$, *where* $\Theta^\star$ *is the set of local Stackelberg equilibria.*

Proof. The two-timescale update forms a stochastic approximation system. By Borkar's multi-timescale convergence theorem [8], the fast variables (x_t, υ_t, z_t) track the lower-level equilibria induced by $(\mathbf{p}_t, \mathbf{n}_t^{\min})$, while the slow variables follow the projected ordinary differential equation (ODE) of the upper-level Stackelberg dynamics. Consequently, the iterates Θ_t converge to a neighborhood of $\Theta^\star$ with deviation $O(\varepsilon)$, which vanishes as the ARMS error $\varepsilon \to 0$. □

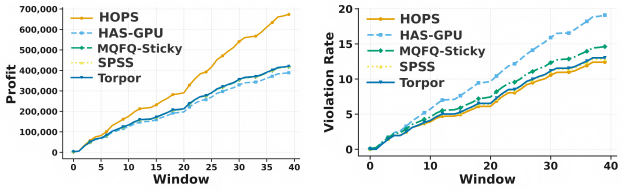
Theorems 1–3 jointly establish that HOPS converges to a stable, near-optimal SE under stochastic dynamics, ensuring adaptive price learning and scalable scheduling stability.

IV. PERFORMANCE EVALUATION

A. Experimental Setup

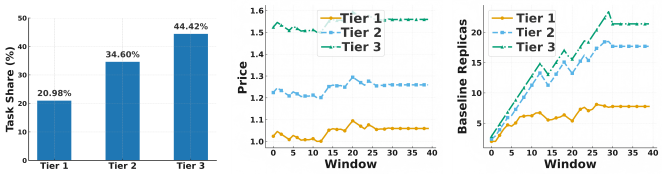
We evaluate HOPS using a trace-driven simulator of a serverless GPU platform with $N = 64$ vGPUs, where each experiment runs for $T = 40$ windows consisting of $H = 60$ slots. The job arrival process is driven by a real-world Azure Functions trace [9], which is aggregated into per-window arrival rates to capture the burstiness and diurnal variations of serverless workloads, and then distributed across three service tiers using the multinomial logit model in (1) to reflect realistic user behaviors. Tier-specific GPU demand g_m, service rate μ_m, and SLA targets s_m are extracted from the Alibaba GPU cluster trace [10], which provides fine-grained information on vGPU usage and AI job characteristics. This combination of realistic invocation patterns (Azure trace) and authentic GPU workload heterogeneity (Alibaba trace) enables faithful reproduction of a multi-tenant serverless GPU environment. All monetary values, including prices, costs, and SLA penalties, are normalized to allow fair comparison among algorithms.

We compare HOPS with four baselines under identical workloads: *(i) Torpor* [11]: a serverless GPU system supporting model swapping and latency-sensitive scheduling; *(ii) HAS-GPU* [6]: a hybrid auto-scaling mechanism with fine-grained GPU slicing for SLO guarantees; *(iii) MQFQ-Sticky* [5]: integrates multi-queue fair scheduling with GPU memory management to reduce cold starts and queuing delays; *(iv) Static Pricing + Separate Scheduling (SPSS)*: an ablated version of our design with fixed prices and baseline allocations to isolate the effect of dynamic pricing.



(a) Profit (b) SLA Violation Rate

Fig. 2: Profitability and SLA compliance with time increasing.



(a) Tier Share (b) Price (c) Replicas

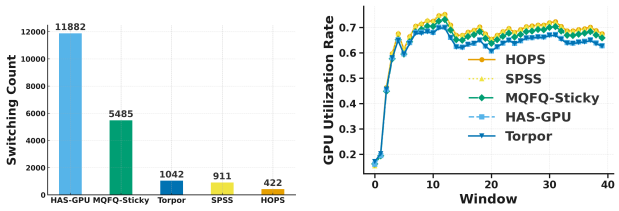
Fig. 3: Convergence of Pricing and Baseline Decisions.

B. Profitability and SLA Compliance

Fig. 2 presents the cumulative profit and SLA violation rate over time across all methods. As shown in Fig. 2(a), HOPS achieves the highest cumulative profit, reaching nearly 6.7×10^5 at the end of the experiment, while the best baseline (HAS-GPU) stays around 4.4×10^5, corresponding to an improvement of roughly 45–55%. The gain stems from HOPS's joint pricing–scheduling mechanism, which continuously adapts service prices and baseline vGPU allocations to workload dynamics, improving utilization and reducing idle capacity. In Fig. 2(b), HOPS maintains a low SLA violation rate, stabilizing around 14% by the 40th window, whereas HAS-GPU shows over 19% violations due to aggressive scaling. MQFQ-Sticky, SPSS, and Torpor exhibit intermediate performance. These results demonstrate that HOPS effectively balances profitability and SLA compliance by jointly optimizing demand shaping and real-time scheduling, avoiding the trade-off typical of decoupled strategies.

C. Convergence of Pricing and Baseline Decisions

Fig. 3 demonstrates the overall convergence behavior of the Pricing and Baseline Adjustment (PBA) process in HOPS. As shown in Fig. 3(a), the final tier distribution stabilizes around 21%, 35%, and 44% for Tiers 1–3, respectively, clearly indicating that the system effectively differentiates user demand across service levels. Fig. 3(b) further shows that prices of all tiers gradually stabilize after about 20 windows, maintaining clear separation among tiers (approximately 1.1, 1.3, and 1.5). This reflects the system's ability to accurately learn users' price elasticity and SLA preferences through stochastic updates. In Fig. 3(c), baseline replicas steadily increase before converging, with Tiers 1–3 settling near 8, 15, and 20 replicas, respectively. The joint stabilization of price and baseline allocation confirms that HOPS achieves equilibrium between economic and operational layers, validating the theoretical Stackelberg stability discussed in Section III-D.



(a) Switching Overhead (b) GPU Utilization

Fig. 4: Switching Overhead and GPU Utilization.

D. Switching Overhead and GPU Utilization

Fig. 4 compares the switching overhead and GPU utilization among all schemes. As shown in Fig. 4(a), HOPS achieves the lowest switching count of 422, less than 4% of HAS-GPU and an order of magnitude smaller than MQFQ-Sticky. This demonstrates that adaptive reconfiguration in HOPS effectively minimizes unnecessary function migrations by learning stable allocation patterns through feedback between the pricing and scheduling layers. Despite this substantial reduction in switching operations, Fig. 4(b) shows that HOPS maintains a consistently high GPU utilization rate (around 0.70–0.72), comparable to or slightly higher than other strong baselines. These results indicate that HOPS achieves efficient and sustainable resource utilization without frequent reallocation, balancing performance stability and operational overhead.

V. Related Work

Serverless GPU scheduling. Recent efforts have extensively optimized GPU allocation and cold-start mitigation for latency-critical serverless workloads. MQFQ-Sticky [5] enhances fairness and GPU memory reuse across queues to reduce cold-start overhead, while HAS-GPU [6] introduces hybrid auto-scaling with fine-grained GPU slicing to satisfy SLO targets under highly heterogeneous workloads. HarmonyBatch [12] further adopts SLO-aware micro-batching to amortize invocation latency. Although these systems significantly improve responsiveness and overall utilization, they only address *how to allocate resources* reactively. The underlying demand remains largely exogenous and uncontrollable, limiting the ability to stabilize utilization and profitability.

Pricing in cloud systems. Cloud pricing mechanisms have been extensively explored in VM-level and spot-instance markets [13], [14], as well as in dynamic resource auctions and game-theoretic optimization [15], [16]. These studies primarily target long-term or coarse-grained resource leasing, assuming relatively stable demand and fixed capacity. Such assumptions are inadequate for second-scale elasticity and SLA-driven GPU services, where resource prices and availability must co-adapt in real time to stochastic workloads.

Our distinction. Unlike prior approaches, our work integrates *economic decision-making (pricing)* with *system-level scheduling* through a unified bi-level framework. By formulating their interaction as a Stackelberg game, the proposed HOPS framework transforms the provider from a passive allocator into an *active demand shaper*, jointly optimizing profitability, SLA compliance, and GPU utilization in dynamic, bursty serverless environments.

VI. Conclusion

This paper proposed HOPS, a hierarchical framework for joint pricing and scheduling optimization in serverless GPU services. By formulating the coordination between pricing and scheduling as a Stackelberg game, HOPS transforms the cloud provider from a passive allocator into an active demand shaper. The upper-level PBA algorithm dynamically tunes prices and baseline configurations, while the lower-level ARMS scheduler ensures efficient GPU utilization and SLA compliance through adaptive reconfiguration. Experimental results on real-world traces validate that HOPS consistently outperforms existing approaches such as HAS-GPU and MQFQ-Sticky in profit, SLA satisfaction, and operational efficiency. Future work will extend HOPS to multi-cluster and heterogeneous GPU environments, exploring online learning mechanisms to further enhance convergence speed and robustness.

References

[1] T. B. Brown, B. Mann *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[2] M. Shahrad, R. Fonseca *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proceedings of the USENIX ATC*, 2020, pp. 205–218.

[3] M. Carlee, W. Zhang, and A. Singh, "Dynamic gpu scheduling for efficient large model inference in cloud platforms," *IEEE Transactions on Cloud Computing*, vol. 12, no. 3, pp. 845–858, 2024.

[4] A. Mazumdar and R. Rao, "Cloud resource pricing and demand shaping: A game-theoretic approach," *ACM Transactions on Economics and Computation*, vol. 11, no. 2, pp. 1–22, 2023.

[5] J. Li, R. Kumar, and T. Yang, "MQFQ-sticky: Multi-queue fair scheduling with gpu memory management for cold-start reduction," in *Proceedings of the 52nd ISCA*. ACM, 2025, pp. 110–121.

[6] H. Gu, R. Zhao, L. Sun, and W. Lin, "HAS-GPU: Hybrid auto-scaling with fine-grained gpu slicing for slo guarantees in serverless platforms," in *Proceedings of the IEEE ICDCS*. IEEE, 2025, pp. 456–468.

[7] D. McFadden, "Conditional logit analysis of qualitative choice behavior," in *Frontiers in Econometrics*, P. Zarembka, Ed. New York: Academic Press, 1974, pp. 105–142.

[8] V. S. Borkar, *Stochastic Approximation: A Dynamical Systems Viewpoint*. Cambridge, UK: Cambridge University Press, 2008.

[9] Microsoft Azure, "Azure functions public dataset," https://github.com/Azure/AzurePublicDataset, 2019, accessed: 2025-09-19.

[10] L. Wang, X. Yu *et al.*, "Characterizing, modeling, and benchmarking gpu datacenter workloads at alibaba," in *Proceedings of the 19th NSDI*. USENIX Association, 2022, pp. 1–17. [Online]. Available: https://github.com/alibaba/clusterdata

[11] X. Yu, L. Zhang, Y. Wang, and K. Chen, "Torpor: A serverless gpu system with model swapping and latency-sensitive scheduling," in *Proceedings of the 52nd ISCA*. ACM, 2025, pp. 123–135.

[12] L. Zhang, X. Yu, and K. Chen, "Harmonybatch: Slo-aware micro-batching for latency-efficient serverless gpu inference," in *Proceedings of the ACM SoCC*. ACM, 2024, pp. 245–258.

[13] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *Proceedings of the IEEE INFOCOM*. IEEE, 2014, pp. 433–441.

[14] W. Shi, L. Zhang *et al.*, "An online auction framework for dynamic resource provisioning in cloud computing," *IEEE/ACM transactions on networking*, vol. 24, no. 4, pp. 2060–2073, 2015.

[15] G. Mencagli, "A game-theoretic approach for elastic distributed data stream processing," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 11, no. 2, pp. 1–34, 2016.

[16] Z. Tang, S. Ren, and Y. Zhang, "Game-theoretic revenue optimization in elastic cloud platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2401–2413, 2022.