Following the Usage, Not the Request: Risk-Aware Task Scheduling with Overbooking in Edge Clouds

Tie Ma[1], Shan Zhang[2*], Xiaoyu Zhang[3], Zichuan Zheng[1], Zhiyuan Wang[2], and Hongbin Luo[2]
[1]School of Computer Science and Engineering, Beihang University, Beijing, China
[2]School of Cyber Science and Technology, Beihang University, Beijing, China
[3]Shen Yuan Honors College, Beihang University, Beijing, China

***Abstract*—Edge computing platforms are increasingly deployed to support delay-sensitive and resource-intensive applications. However, current task scheduling strategies, which rely on user-requested resources, often lead to low resource utilization and reduced platform profit due to users' tendency to over-request resources. Overbooking, widely adopted in industries such as airlines and hotels, can improve utilization but introduce risk under uncertain task resource usage. This paper applies resource overbooking to edge clouds, focusing on task scheduling optimization under uncertainty. We formulate the problem as a stochastic mixed integer program, which is proven to be NP-hard. To this end, a risk evaluation scheme is proposed to accurately quantify the risk of overload without assuming specific resource usage distributions, which has an additive error guarantee. Based on this scheme, we transform the problem into a more deterministic form and prove that the objective function is submodular. This enables us to design a greedy algorithm that achieves a $(1-1/e)$-approximation ratio with lower complexity. Extensive experiments on a real-world dataset demonstrate that the proposed algorithm significantly improves profit by 0.23×-3.35× and resource utilization by 0.16×-0.75× while accurately controlling the risk associated with overbooking.**

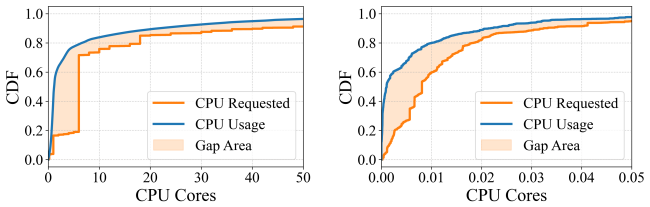
I. INTRODUCTION

A. Background and Motivation

We have witnessed the growing popularity of edge computing platforms, which enable users to run delay-sensitive and resource-intensive applications from the edge of the networks. The providers (e.g., Google [1], Amazon [2], and Alibaba [3]) of these platforms build a geographically distributed and massive infrastructure, setting up hundreds of edge nodes and connecting them via the backhaul network.

Edge computing platforms have matured into an operational paradigm that appears straightforward from the user's perspective: users simply submit tasks with a requested amount of resources to a nearby edge node, and the platform promptly returns the results. However, the underlying complexity is significant from the platform's standpoint. Strategic task scheduling is a critical issue to optimize key objectives (e.g., profit, resource utilization) with a resource pool shared among users.

This work was supported in part by the National Key R&D Program of China under Grant 2022YFB4501000, in part by the Nature Science Foundation of China under Grant 624B2015, 62422201, 62271019, 62225201, U24B20128, in part by the Fundamental Research Funds for the Central Universities, China, and State Key Laboratory of Complex & Critical Software Environment. Corresponding author: Shan Zhang.



(a) Alibaba Dataset (b) Google Dataset

Fig. 1. Gap analysis between requested and actual resource usage. The analysis is conducted based on the prototype from [4].

While intuitively assigning tasks to the nearest edge node can reduce execution latency, this naive approach may result in resource hotspots or even task failures if the local node lacks sufficient resources. Consequently, it is imperative for the platform to make task scheduling decisions in a holistic and adaptive manner.

The above paradigm is typically based on the resources *requested* by users [5], rather than their actual resource *usage*. However, a substantial gap exists between the resource requested and the resource usage in practice. For instance, our analysis of two real-world datasets from Alibaba [4] and Google [6] reveals that only 43% and 48% of the requested CPU resources are actually consumed, respectively, as shown in Fig. 1. This gap is primarily due to the difficulty users face in accurately estimating their resource needs, leading them to routinely over-request resources [7]–[9]. As a result, production clusters such as those at Google suffer from a low CPU utilization ranging from 20% to 35% [10]. This underutilization not only reduces the platform's profit but also forces more user tasks to be offloaded to other remote clouds, leading to increased latency and operational costs.

To address similar challenges, various commercial sectors, such as hotels [11] and airlines [12], adopt *overbooking* to sell more resources than are available, thereby improving their resource utilization. For example, airlines routinely overbook flights to compensate for passenger no-shows, thereby maximizing profit and maintaining high seat utilization; without overbooking, more than 15% of seats may remain empty, resulting in substantial economic losses [13], [14]. This motivates us to apply overbooking to the edge clouds. However, overbooking also introduces risk. If an edge node becomes

overloaded, tasks may miss their deadlines, and the platform's profit can decline. Controlling this risk is challenging, as task resource usage is inherently *uncertain* and only the historical distributions are available prior to scheduling. This leads to the first key question in this paper:

Q1. *How to quantify and control the risk (with uncertainty characteristics) incurred by overbooking?*

One feasible way is to design a trading mechanism between users and the platform based on contract theory (e.g. [14]–[18]). However, this approach ignores the uncertainty of the resource usage and only considers the no-show scenario, which means the users may request the resources but do not execute their tasks. This coarse-grained method often leads to suboptimal resource utilization in practice.

A natural idea to tackle the uncertainty is to quantify and analyze the probability of overloading with respect to the overbooking strategies. However, this quantification is non-trivial, as it involves complex convolution operations and cannot express the risk in a closed form. In fact, even computing the exact overload probability for a given task set on a given edge node is NP-hard [19]. Only in special cases, such as when task resource usage follows a specific distribution (e.g., Normal distribution), can the overload probability be computed efficiently. Following such a design philosophy, many existing studies (e.g., [20], [21]) simplify the problem under the assumption that the resource usage of tasks follows a specific distribution. However, the resource usage of different tasks may present distinct features in practice, and we cannot always obtain the explicit-form overloading probability. This leads to our second key question in this paper:

Q2. *How to achieve a performance-guaranteed task scheduling without assuming specific resource usage distributions?*

Wu et al. [22] leverages the Γ-robustness theory to address this challenge, which still relies on the assumption that the distribution follows a symmetric feature. Recently, learning-based approaches (e.g., [23]) have been proposed to address the implicit distribution challenge; however, they lack interpretability and a theoretical guarantee. The overloading risk cannot be concisely captured based on each task's resource usage, posing a significant challenge to designing performance-guaranteed scheduling algorithms. Moreover, the constraints introduced by the spatial cooperation among edge nodes further complicate the problem. To the best of our knowledge, Q2 remains an open challenge in the existing literature.

B. Main Results and Key Contributions

This paper investigates the resource overbooking problem in edge clouds, with a focus on task scheduling optimization. Specifically, users submit tasks to a shared resource pool, and the platform must schedule these tasks based on the current status of edge node capacities and inter-node transmission costs. In practice, task resource usage is uncertain, and only historical distributions are available. To maximize profit while maintaining low risk, the platform must effectively address this uncertainty.

Our main results and key contributions are as follows:

- *Novel Problem Formulation:* We study the task scheduling problem in edge clouds with resource overbooking, aiming to maximize the platform's profit while controlling the risk of resource overload. The problem is formulated as a Stochastic Mixed Integer Programming (SMIP) problem and proven to be NP-hard by reduction from the classic multiple knapsack problem.
- *Design of Risk Evaluation Scheme:* With a properly constructed utility function, we transform the risk evaluation problem into a high-dimensional space to decompose the contribution of each task along individual dimensions in an additive manner. Then, an Expected Utility-based Risk Evaluation scheme (EURE) is proposed, which can precisely quantify the risk of scheduling any task set on a node with a slightly relaxed capacity. EURE provides an additive error bound ϵ for any constant $\epsilon > 0$, without assuming specific distribution forms or features.
- *Scheduling Algorithm Design:* The EURE scheme enables us to transform the original SMIP problem into a more tractable deterministic form, whose objective function is rigorously proved to be submodular. Building on this theoretical foundation, we design a risk-aware task scheduling algorithm that efficiently solves the problem using a greedy approach. Our analysis shows the proposed algorithm achieves a $(1-1/e)$-approximation ratio.
- *Extensive Evaluation:* Comprehensive experiments are conducted based on real-world datasets, and the results demonstrate that the proposed algorithm improves the profit by 0.23×-3.35× over state-of-the-art approaches while improving the resource utilization by 0.16×-0.75×. Furthermore, simulations confirm the accuracy of our algorithm in controlling the risk of resource overbooking.

II. System Model and Problem Formulation

This section presents the system model and mathematical formulation of the edge cloud resource overbooking problem.

A. Task Model

We consider a typical edge cloud scenario consisting of N connected edge nodes denoted by $\mathcal{N} = \{1, 2, \cdots, n, \cdots, N\}$, as shown in Fig. 2. Each edge node is equipped with a certain amount of computing resource, denoted by C_n where $n \in \mathcal{N}$. Users generate and send requests to the nearest edge nodes for potential computing services. Denote by $\mathrm{R} = [R_1, R_2, \cdots, R_N]$ the number of requests received at each edge node. The requests are of K types. A request of type-k requires b_k computing resources in booking, and yields a revenue of p_k if successfully served. The actual resource usage is uncertain and can be modeled by a random variable u_k following a known probability distribution. Generally, u_k cannot exceed r_k according to practical business rules. In addition, the resource usage of each request is considered to be independent. Note that although we focus on computing resources, the

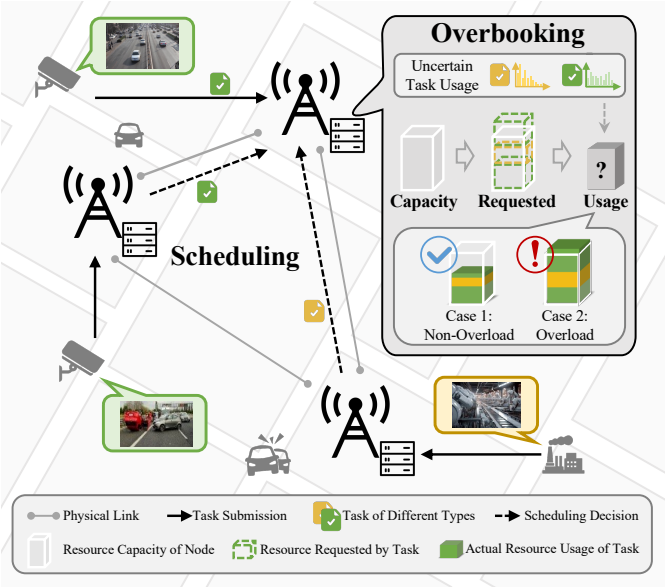


Fig. 2. System model overview.

model can be readily extended to other resource types during task execution (e.g., memory and energy consumption).

B. Scheduling Model

Let $x_{r,n,m}$ indicates if the r-th request of node n is served by node m or not, where $r = 1, 2, \cdots, R_n$, $n \in \mathcal{N}$ and $m \in \mathcal{N}$. Specifically, the corresponding tasks will be served in three cases: (1) routed to the remote cloud for service if $\sum_{m\in\mathcal{N}} x_{r,n,m} = 0$, (2) served locally if $x_{r,n,n} = 1$, and (3) offloaded to other edge nodes if $x_{r,n,m} = 1$ and $m \neq n$. $x_{r,n,m}$ directly influence the system costs. The third case introduces a cooperation cost $t_{r,n,m}$ regarding the fronthaul transmissions. We have $t_{r,n,m} = d_{r,n}\tau_{n,m}$, where $d_{r,n}$ is the size of input/output data of r-th request at node n and $d_{r,n}$ is the normalized transmission cost depending on the distance and bandwidth conditions between nodes n and m.

An edge node becomes overload when its equipped resources cannot meet the usage of all scheduled tasks. Let δ_n be a 0-1 indicator to show if node n is overloaded. Specifically, δ_n is given by:

$$\delta_n = \begin{cases} 1, & \text{if } \frac{\sum_{m\in\mathcal{N}}\sum_{r=1}^{R_m} x_{r,m,n}u_{k_{r,m}}}{C_n} \leq 1, \\ 0, & \text{if } \frac{\sum_{m\in\mathcal{N}}\sum_{r=1}^{R_m} x_{r,m,n}u_{k_{r,m}}}{C_n} > 1, \end{cases} \tag{1}$$

where $k_{r,m}$ is the type of task corresponding to the r-th request received at node m. Due to the limited budget of overload cost, the platform needs to ensure that the probability of $\delta_n = 0$ is below a certain threshold Δ, i.e., $\Pr(\delta_n = 1) \leq \Delta$.

C. Problem Formulation

The profit of a scheduled task is its revenue minus its cooperation cost. Our objective is to maximize the total profit under the predefined overload risk constraints. The problem

TABLE I
LIST OF NOTATIONS

Symbol	Definition
$\mathcal{N}$	Set of edge nodes
N	Number of edge nodes
C_n	Computing resource capacity of node n
R_n	Number of requests received at node n
K	Number of task types
$k_{r,n}$	Type of the r-th request at node n
p_k	Revenue of a type-k task
b_k	Requested resource demand of a type-k task
u_k	Actual resource usage of a type-k task
$x_{r,n,m}$	Binary scheduling variable indicating whether the r-th request of node n is served by node m
$t_{r,n,m}$	Cooperation cost for serving the r-th request of node n at node m
$d_{r,n}$	Size of input/output data of the r-th request at node n
$\tau_{n,m}$	Normalized transmission cost between nodes n and m
δ_n	Binary indicator for whether node n is overloaded
Δ	Threshold for overload probability constraint

can be formulated as the following Stochastic Mixed Integer Program (SMIP):

$$\begin{aligned} (\mathbf{P1})\ \max & \sum_{n\in\mathcal{N}}\sum_{m\in\mathcal{N}}\sum_{r=1}^{R_n} x_{r,n,m}(p_{k_{r,n}} - t_{r,n,m}) \\ \text{s.t.}\quad & x_{r,n,m} \in \{0,1\}, \quad \forall r = 1,\cdots,R_n, n \in \mathcal{N}, m \in \mathcal{N}, \\ & \sum_{m\in\mathcal{N}} x_{r,n,m} \leq 1, \quad \forall r = 1,\cdots,R_n, n \in \mathcal{N}, \\ & \Pr(\delta_n = 0) \leq \Delta, \quad \forall n \in \mathcal{N}. \end{aligned} \tag{2}$$

The first constraint ensures that the task scheduling variables are valid. The second constraint ensures that each task is scheduled to at most one edge node. The last chance constraints realize the risk control at each edge node.

This problem is complex since the chance constraints involve the sum of random variables, which requires convolution operations and cannot be expressed in a closed form. Moreover, the problem remains NP-hard even when the values of u_k are deterministic, as shown in the following theorem.

Theorem 1. *Problem (**P1**) is NP-hard.*

Proof. Consider a subproblem of **P1** obtained by treating u_k as deterministic constants rather than stochastic variables. Then, the problem reduces to the classical 0-1 multi-knapsack problem (MKP), where tasks are treated as items and edge nodes as knapsacks. Since MKP is NP-hard and it is a subproblem of **P1**, solving **P1** is at least as hard as solving the 0-1 multi-knapsack problem; hence, **P1** is NP-hard. □

III. ALGORITHM DESIGN

In this section, we present the design of the proposed Risk-aware Task Scheduling algorithm based on Submodularity (RTSS) for solving the SMIP problem. The overall framework is illustrated in Fig. 3. RTSS consists of two main components: *Risk Evaluation* (§III-A) and *Submodular Scheduling* (§III-B). The Risk Evaluation component quantifies the overload risk of scheduling a given task set on a node. It transforms the risk evaluation problem into a high-dimensional space,

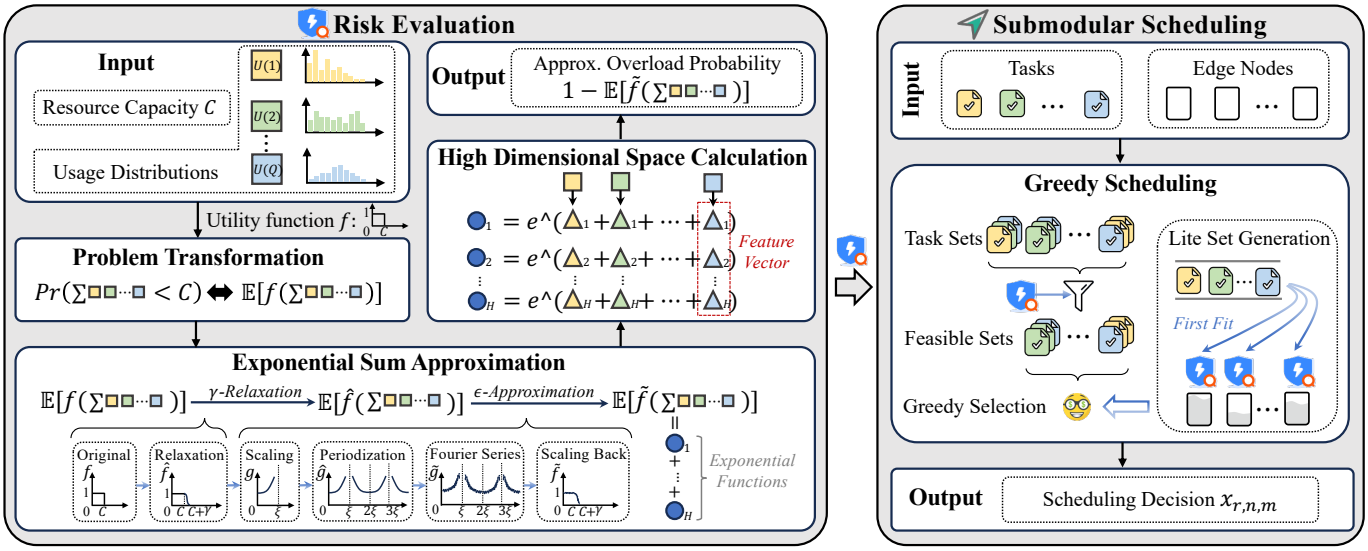


Fig. 3. Overview of the RTSS algorithm.

where each task's contribution to the overload probability can be decomposed into an additive form. The Submodular Scheduling component leverages the Risk Evaluation results to reformulate the original SMIP problem into a tractable form that can be efficiently solved using submodular optimization techniques. We provide a rigorous theoretical analysis of the performance guarantees of both components.

A. Expected Utility-based Risk Evaluation

The overload constraints pose a significant challenge in problem (**P1**), as they involve inherent uncertainty and require computationally complex convolution operations. To address this constraint, we focus on the following fundamental question in this subsection: *Given a set of tasks and an edge node, how to evaluate the risk of exceeding the resource capacity if the task set is scheduled to the node?*

Inspired by signal processing techniques that transform time domain convolution operations into frequency-domain multiplication operations, our key idea is to leverage expected utility theory [24], [25] to transform the risk evaluation problem into a high-dimensional space and decompose the resource usage of each task in this space.

We introduce some notations to simplify the presentation in this subsection. Denote by the task set as $\mathcal{Q} = \{1, 2, \cdots, q, \cdots, Q\}$, where Q represents the total number of tasks to schedule. We denote the resource usage of task q as $U(q)$ and the aggregate resource usage of task set $\mathcal{Q}$ as $U(\mathcal{Q})$. Let C denote the resource capacity of the edge node.

We first construct a utility function f:

$$f(U(\mathcal{Q})) = \begin{cases} 1, U(\mathcal{Q}) \in [0, C], \\ 0, U(\mathcal{Q}) > C, \end{cases} \tag{3}$$

whose expected value equals to the non-overload probability of the edge node, i.e. $\mathbb{E}[f(U(\mathcal{Q}))] = \Pr(U(\mathcal{Q}) \leq C)$.

The following lemma motivates us to use a series of exponential functions to approximate the utility function f, whereby the impact of each task can be decomposed.

Lemma 1. *If a utility function $\tilde{f}$ is defined as a sum of exponential functions, i.e., $\tilde{f}(U(\mathcal{Q})) = \sum_{i=1}^{H} l_i b_i^{U(\mathcal{Q})}$, where H is the number of terms, l_i is the coefficient, and b_i is the base, then the expected value of $\tilde{f}(U(\mathcal{Q}))$ can be expressed as follows:*

$$\mathbb{E}[\tilde{f}(U(\mathcal{Q}))] = \sum_{i=1}^{H} l_i e^{\sum_{q\in Q} \log \mathbb{E}[b_i^{U(q)}]}. \tag{4}$$

Proof. For each exponential function $b_i^{U(\mathcal{Q})}$, we have:

$$\mathbb{E}[b_i^{U(\mathcal{Q})}] = \mathbb{E}[b_i^{\sum_{q\in Q} U(q)}] = \mathbb{E}\left[\prod_{q\in Q} b_i^{U(q)}\right] = \prod_{q\in Q} \mathbb{E}[b_i^{U(q)}]. \tag{5}$$

By taking the logarithm of both sides, we have:

$$\log \mathbb{E}[b_i^{U(\mathcal{Q})}] = \sum_{q\in Q} \log \mathbb{E}[b_i^{U(q)}]. \tag{6}$$

Combine Eq. (5) and Eq. (6), we can rewrite $\mathbb{E}[\tilde{f}(U(\mathcal{Q}))]$ as follows:

$$\mathbb{E}[\tilde{f}(U(\mathcal{Q}))] = \mathbb{E}[\sum_{i=1}^{H} l_i b_i^{U(\mathcal{Q})}] = \sum_{i=1}^{H} l_i e^{\sum_{q\in Q} \log \mathbb{E}[b_i^{U(q)}]}. \tag{7}$$

□

Lemma 1 motivates us to utilize *Fourier series* to approximate the utility function f. However, the Fourier series should apply to periodic functions, while the expectation of the periodic version of f is not equivalent to $\Pr(U(\mathcal{Q}) \leq C)$ anymore. To bridge this gap, we employ a scaling and scaling back approach as [24], based on the key insight that the exponential function maintains large values even for small arguments. Specifically, we first multiply the utility function by an exponential term to scale up, then shift the resulting product within its domain of definition to construct a periodic

Algorithm 1: Expected Utility-based Risk Evaluation (EURE)

Input: A task set $\mathcal{Q}$, the resource capacity C of a edge node;
Output: Overload probability if $\mathcal{Q}$ is scheduled to the edge node

Step 1: Feature Vector Calculation
Calculate the feature vector $\mathbf{v}_q$ of each task according to Eq. (9);
Step 2: Summation
Initialize the sum of feature vector $\mathbf{V} = \mathbf{0}$;
for *each task q in Q* **do**
  $\mathbf{V} \leftarrow \mathbf{V} + \mathbf{v}_q$;
Step 3: Risk Evaluation
Calculate the overload probability based on Eq. (4);

function. After performing the Fourier series approximation on this periodic function, we divide the result by the original exponential multiplier to scale back. As such, the utility function is restored to its original form within the desired domain, while the contributions from shifted regions become negligible due to division by exponentially large values. This transform technique effectively enables Fourier series approximation of the utility function while preserving the approximately equivalence to the original problem $\Pr(U(\mathcal{Q}) \leq C)$.

As illustrated in the lower left corner of Fig. 3, this transformation process consists of the following five steps: *i) Relaxation:* To constrain the number of terms in the Fourier series, we slightly relax the utility function f into $\hat{f}$ given by:

$$\hat{f}(U(\mathcal{Q})) = \begin{cases} 1, & |U(\mathcal{Q})| \in [0, C], \\ -\dfrac{|U(\mathcal{Q})|}{\gamma} + \dfrac{C}{\gamma} + 1, & |U(\mathcal{Q})| \in [C, C+\gamma], \\ 0, & |U(\mathcal{Q})| > C + \gamma, \end{cases} \tag{8}$$

where $\gamma > 0$ is an arbitrary positive constant. This step turns the problem into a relaxed version, i.e., $\Pr(U(\mathcal{Q}) \leq C + \gamma)$, which is close to the original problem and easy to handle, but introduces a small additive error γ. Note that only the nonnegative part of $U(\mathcal{Q})$ impacts the final result, and $\hat{f}(U(\mathcal{Q}))$ can take other forms that satisfy the α-Hölder condition [24]. *ii) Scaling:* We transform $\hat{f}(U(\mathcal{Q}))$ into a scaled version $g(U(\mathcal{Q})) = \eta^{U(\mathcal{Q})}\hat{f}(U(\mathcal{Q}))$ in the domain $U(\mathcal{Q}) \in [-\xi, \xi]$, where ξ and η are constants that can be set according to [24]. *iii) Periodization:* We then construct a periodic function $\hat{g}(U(\mathcal{Q}))$ by replicating the scaled function $g(U(\mathcal{Q}))$ defined on $[-\xi, \xi]$ across adjacent intervals. *iv) Fourier series:* By applying Fourier series to $\hat{g}$, we get $\tilde{g}(U(\mathcal{Q})) = \sum_{i=1}^{H} l_i \phi_i^{U(\mathcal{Q})}$, where H is the number of basis functions, ϕ_i is the i-th base, and l_i is the corresponding coefficient. *v) Scaling back:* Let $b_i = \phi_i/\eta$, we scale back $\tilde{g}(U(\mathcal{Q}))$ in the domain $U(\mathcal{Q}) \in [0, \infty]$ and get the final approximation $\tilde{f}(U(\mathcal{Q})) = \sum_{i=1}^{H} l_i(\phi_i/\eta)^{U(\mathcal{Q})} = \sum_{i=1}^{H} l_i b_i^{U(\mathcal{Q})}$.

Lemma 2. *$\tilde{f}(U(\mathcal{Q})) = \sum_{i=1}^{H} l_i b_i^{U(\mathcal{Q})}$ is an ϵ-approximation of $\hat{f}(U(\mathcal{Q}))$, i.e., $|\tilde{f}(U(\mathcal{Q})) - \hat{f}(U(\mathcal{Q}))| \leq \epsilon$, where ϵ is an arbitrary positive constant and H is a small value only depending on ϵ.*

Proof. The proof is similar to the proof of Theorem 2 in [25], thus we omit the details here. Note that H can be determined based on Corollary 1 in [25]. □

Lemma 2 shows that we concisely approximate the slightly relaxed utility function $\hat{f}$ with a sum of exponential functions which forms a H-dimensional space. In the H-dimensional space, we can tackle each task's contribution in an additive form by using *feature vector*.

Definition 1. *The feature vector of task q is defined as:*

$$\mathbf{v}_q = \langle \log \mathbb{E}[b_1^{U(q)}], \log \mathbb{E}[b_2^{U(q)}], \cdots, \log \mathbb{E}[b_H^{U(q)}] \rangle. \tag{9}$$

We denote the sum of the feature vectors of the task set $\mathcal{Q}$ as $\mathbf{V} = \sum_{q \in Q} \mathbf{v}_q$. The additive property of V is quite interesting, which can ease the calculation of non-overload probability by performing an incremental update of V according to Eq. (4). In this regard, we propose an Expected Utility-based Risk Evaluation algorithm (EURE), which is summarized in Algorithm 1. In what follows, we analyze how accurately the EURE algorithm can evaluate the overload risk.

Theorem 2. Given a task set $\mathcal{Q}$ and a resource capacity C, the EURE algorithm can calculate the probability of exceeding the relaxed capacity $C + \gamma$ with an additive error ϵ, i.e., $|\mathbb{E}[\tilde{f}(U(\mathcal{Q}))] - \Pr(U(\mathcal{Q}) \leq C + \gamma)| \leq C\epsilon + \gamma$, where γ and ϵ are arbitrary positive constants.

Proof. According to the definition of expectation in measure theory [26], we have:

$$\begin{aligned} &\left|\mathbb{E}[\tilde{f}(U(\mathcal{Q}))] - \Pr(U(\mathcal{Q}) \leq C + \gamma)\right| \\ &\leq \left|\mathbb{E}[\tilde{f}(U(\mathcal{Q}))] - \mathbb{E}[\hat{f}(U(\mathcal{Q}))]\right| \\ &+ \left|\mathbb{E}[\hat{f}(U(\mathcal{Q}))] - \Pr(U(\mathcal{Q}) \leq C + \gamma)\right| \\ &= \left|\int (\tilde{f}(U(\mathcal{Q})) - \hat{f}(U(\mathcal{Q}))) \, dP_Q(U(\mathcal{Q}))\right| + \gamma, \end{aligned} \tag{10}$$

where P_Q denotes the probability measure of $U(\mathcal{Q})$. From Lemma 2, we know that $|\tilde{f}(U(\mathcal{Q})) - \hat{f}(U(\mathcal{Q}))| \leq \epsilon$ for all $U(\mathcal{Q})$. Thus we can bound the first term of Eq. (10) as follows:

$$\begin{aligned} &\left|\int (\tilde{f}(U(\mathcal{Q})) - f(U(\mathcal{Q}))) \, dP_Q(U(\mathcal{Q}))\right| \\ &\leq \left|\int \epsilon \, dP_Q(U(\mathcal{Q}))\right| \leq C\epsilon. \end{aligned} \tag{11}$$

This completes the proof. □

Theorem 2 shows that the proposed EURE algorithm provides a rigorous ϵ-approximation to the slightly relaxed overload risk. The complexity of the EURE algorithm is $O(QH)$, composed of the following parts: (1) Calculating v_q requires $O(H)$ for each task; (2) Summing up the feature vectors requires $O(QH)$; (3) Calculating the expected utility based on Eq. (4) requires $O(H)$. Note that the EURE algorithm involves

vector operations, which can be efficiently implemented using matrix operations [27] in practice.

B. Submodularity-based Task Scheduling

Based on Algorithm 1, we can transform the problem (**P1**) into a more deterministic form:

$$
\begin{aligned}
(\mathbf{P2}) \max & \sum_{n\in\mathcal{N}}\sum_{m\in\mathcal{N}}\sum_{r=1}^{R_n} x_{r,n,m}(p_{k_{r,n}} - t_{r,n,m}) \\
\text{s.t.} \quad & x_{r,n,m} \in \{0,1\}, \quad \forall r = 1,\cdots,R_n, n \in \mathcal{N}, m \in \mathcal{N}, \\
& \sum_{m\in\mathcal{N}} x_{r,n,m} \le 1, \quad \forall r = 1,\cdots,R_n, n \in \mathcal{N}, \\
& EURE(Q_n, C_n) \le \Delta, \quad \forall n \in \mathcal{N}.
\end{aligned} \tag{12}
$$

where $EURE$ is Algorithm 1 that proposed in §III-A, and $Q_n = \{q_{r,m} | x_{r,m,n} = 1, r = 1,\cdots,R_m, m \in \mathcal{N}\}$. $EURE$ provides a way to accurately evaluate the risk of scheduling a task set on a node, which enables us to focus on the *schedulable task sets* and further reformulate the problem (**P2**) into a submodular function maximization problem. For ease of presentation, We define the set of all tasks in the system as $Z = \{q_{r,n} | \forall r = 1,\cdots,R_n, \forall n \in \mathcal{N}\}$.

Definition 2. *The schedulable task set $\Upsilon_n \in Z$ for node n, is defined as the task set that satisfies the following conditions:*

- *The scheduling of any task $q_{r,m}$ in Υ_n yields a positive profit, i.e., $p_{k_{r,m}} > t_{r,m,n}$.*
- *Scheduling all tasks in Υ_n to node n does not violate the risk constraint.*

Definition 3. *Given a collection $\mathcal{S}$, whose elements are disjoint subsets of Z, the objective function of problem (**P2**) can be equivalently formulated as a set function A defined by:*

$$
A(\mathcal{S}) = \sum_{S_i\in\mathcal{S}} \sum_{q_{r,m}\in S_i} \left(p_{k_{r,m}} - t_{r,m,n_i}\right), \tag{13}
$$

where $S_i \in \mathcal{S}$ denotes the i-th subset, t_{r,m,n_i} denotes the minimal cooperation cost incurred when scheduling the tasks in S_i, as determined by Algorithm 2.

Before we prove the submodularity of the set function A, we give a definition of submodularity as follows:

Definition 4. (Nonnegativity, Monotonicity, and Submodularity) *Given a non-empty finite ground set G, a real-valued set function $y : 2^G \to R$, y is called nonnegative, monotone, and submodular if it satisfies the following conditions, respectively:*

- $y(\emptyset) = 0$ and $y(J) \ge 0$ for all $J \subseteq G$ (nonnegative).
- $y(J') \le y(J)$ for all $J' \subseteq J \subseteq G$ (monotone).
- $y(J' \cup e) - y(J') \ge y(J \cup e) - y(J)$, for any $J' \subseteq J \subseteq G$ and $e \in G - J$ (submodular).

Then the submodularity can be proven as follows:

Theorem 3. The set function A defined in Eq. (13) is nonnegative, monotone, and submodular.

Proof. According to the definition of A in Eq. (13), and noting that tasks yielding negative profit are not scheduled, it follows that A is *nonnegative* and *monotone*. We now prove that A is *submodular*. Consider G is the power set of all tasks, i.e., $G = 2^Z$. Consider an arbitrary set of tasks $J \subseteq G$ and an arbitrary set of tasks $B \subseteq G - J$. We have:

$$
A(J \cup B) - A(J) = \sum_{q_{r,m}\in B} (p_{k_{r,m}} - t_{r,m,n_i}), \tag{14}
$$

where t_{r,m,n_i} is the minimal cooperation cost of task $q_{r,m} \in B$ after scheduling tasks in J. Given an arbitrary subset $J' \subseteq J$, we have:

$$
A(J' \cup B) - A(J') = \sum_{q_{r,m}\in B} (p_{k_{r,m}} - t_{r,m,n'_i}), \tag{15}
$$

where t_{r,m,n'_i} is the minimal cooperation cost of task $q_{r,m} \in B$ after scheduling tasks in J'. Note that two cases can happen:

- *Case 1:* The task sets in $J - J'$ do not affect the scheduling of tasks in B. In this case, tasks in B will be scheduled to the same node n, i.e., $t_{r,m,n_i} = t_{r,m,n'_i}$;
- *Case 2:* After scheduling tasks in $J - J'$, node n will violate risk constraint if all tasks in B are scheduled to it, thus we need to schedule some tasks in B to other nodes with larger cooperation costs, i.e., $t_{r,m,n_i} > t_{r,m,n'_i}$.

Therefore, we have:

$$
t_{r,m,n_i} \ge t_{r,m,n'_i}. \tag{16}
$$

Combining Eq. (14), (15) and (16), we have:

$$
A(J \cup B) - A(J) \le A(J' \cup B) - A(J'). \tag{17}
$$

Thus, A is a submodular set function on G. □

Theorem 3 allows us to greedily schedule tasks at the set granularity. Furthermore, Algorithm 1 supports incremental update of the overload risk, allowing for a lightweight task set generation step at the task granularity. This step is particularly suitable for resource-constrained environments, where the generated set can serve as a base and be opportunistically refined to further enhance the scheduling outcome.

Based on the above idea, we propose a Risk-aware Task Scheduling algorithm based on Submodularity (RTSS), and the details are described in Algorithm 2. The first step of the algorithm is to perform a greedy scheduling based on the submodularity (Line 1-13). At the beginning, we initialize the set of schedulable tasks Υ_n for each node n based on Algorithm 1. Then, in each iteration, the algorithm evaluates the potential increase in the function A by adding each schedulable task set S_n to the current set of task sets $\mathcal{S}$. The task set that yields the maximum increase in A is selected and added to $\mathcal{S}$. This process continues until all nodes are filled with tasks or no more tasks can be added without violating the risk constraint. In the second step, we employ a lightweight procedure to rapidly generate base task sets (Line 14-24). Specifically, tasks are sorted according to descending order of their *profit density*, which is defined as $\frac{p_{k_{r,m}}}{\mathbb{E}[u_{k_{r,m}}] + \sqrt{\mathrm{Var}(u_{k_{r,m}})}}$, where $\mathbb{E}[u_{k_{r,m}}]$ and $\mathrm{Var}(u_{k_{r,m}})$ are the expected value and variance of the resource usage of task $q_{r,m}$, respectively. Each task is then assigned to nodes in a first-fit manner until the risk threshold is reached. In the final step (Line 25-28), we compare

Algorithm 2: Risk-aware task scheduling based on submodularity (RTSS)

Input: The set of tasks $q_{r,m}$, the set of nodes n, task profit p_k, cooperation cost $t_{r,n,m}$
Output: Scheduling decision $x_{r,n,m}$ of each task

Step 1: Greedy Scheduling
Initialize the set $\mathcal{S} \leftarrow \emptyset$
Calculate the Schedulable task set Υ_n for each node n based on Algorithm 1
while $|\mathcal{S}| < |\mathcal{N}|$ **do**
  Set $opt \leftarrow 0$, $tmp \leftarrow 0$
  for $n \in \mathcal{N}$ **do**
    for $S_n \in \Upsilon_n - \mathcal{S}$ **do**
      $tmp \leftarrow A(\mathcal{S} \cup \{S_n\})$
      if $tmp > opt$ **then**
        $opt \leftarrow tmp$, $S^* \leftarrow S_n$
  $\mathcal{S} \leftarrow \mathcal{S} + \{S^*\}$
  Update Υ_n based on Algorithm 1
Step 2: Lightweight Set Generation
Initialize the set $S'_n \leftarrow \emptyset$ for each node n
Sort tasks according to profit density
Sort nodes for each task $q_{r,m}$ according to $t_{r,m,n}$
for *each sorted task* $q_{r,m}$ **do**
  for *each sorted node* n **do**
    if $EURE(S'_n \cup \{q_{r,m}\}) \leq \Delta$ **then**
      $\mathcal{S}'_n \leftarrow \mathcal{S}'_n \cup \{q_{r,m}\}$
      break
$\mathcal{S}' \leftarrow \{S'_1, S'_2, ..., S'_{|\mathcal{N}|}\}$
Step 3: Result Selection
if $A(\mathcal{S}') > A(\mathcal{S})$ **then**
  $\mathcal{S} \leftarrow \mathcal{S}'$
Set $x_{r,n,m}$ according to $\mathcal{S}$

the profit yielded by the previous two steps, and choose the one that yields the largest profit as the final scheduling result.

Next, we theoretically analyze the performance of the proposed RTSS algorithm.

Theorem 4. Our proposed algorithm RTSS has an approximation ratio of $(1 - 1/e)$ for the problem (**P2**).

Proof. From theorem 3, we know that the set function $A(\mathcal{S})$ is nonnegative, monotone, and submodular. Meanwhile, the constraint that $|\mathcal{S}| \leq |\mathcal{N}|$ is a cardinality constraint. Due to these properties, the result of the greedy scheduling step in Algorithm 2 has an approximation ratio of $(1 - 1/e)$ [28]. Since the lightweight set generation step does not decrease the value of the function A, the overall approximation ratio of the algorithm RTSS remains $(1 - 1/e)$. □

While the number of schedulable task sets may be exponential, prior work [29] shows that a polynomial number suffices for optimization. To achieve the trade-off between

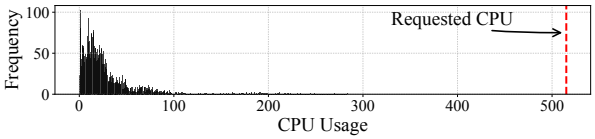


Fig. 4. An example of CPU usage and requested CPU in the real-world trace.

algorithm complexity and performance, we construct only $O(|Z|)$ schedulable task sets per node, where $|Z|$ is the number of tasks in the system. Thus, the time complexity of Step 1 is $O(|\mathcal{N}|^2|Z| + |\mathcal{N}||Z|^2H)$, where the first term corresponds to evaluating the submodular function A for each schedulable task set, and the second term accounts for evaluating $O(|Z|)$ schedulable task sets per node. For Step 2, after sorting tasks and nodes, we iterate through all task-node pairs, updating feature vectors and calculating overload risk via Eq. (4) in $O(H)$ per iteration. Thus, the overall time complexity of Step 2 is $O(|Z|\log|Z| + |\mathcal{N}|\log|\mathcal{N}| + O(|\mathcal{N}||Z|H))$, which is dominated by Step 1. Therefore, the overall time complexity of Algorithm 2 is $O(|\mathcal{N}|^2|Z| + |\mathcal{N}||Z|^2H)$. We reiterate that RTSS can be efficiently implemented using matrix operations [27] in practice.

IV. EVALUATION

In this section, we carry out extensive simulations based on the real-world data trace to validate the performance of the proposed RTSS algorithm.

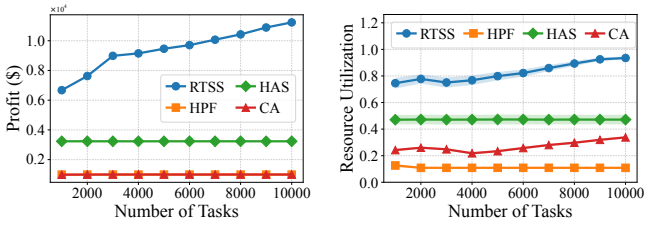
A. Simulation Setup

We conduct our experiment based on the real-world trace [4] from a large production cluster of Alibaba's artificial intelligence platform, as shown in Fig. 4. In this experiment, we focus on the CPU resources. Unless specified otherwise, the default experiment parameters are as follows. From the trace [4], we extract type, data size, CPU usage distribution, and requested CPU of tasks. According to the trace, the CPU capacity of edge servers is set to 64 or 96 cores. The number of tasks is set to 5000, and the number of servers is set to 10. The revenue is set to 1.2$ for each requested CPU core according to Google Cloud Platform [30]. Following the approach in [23], we randomly generated the number of hops between any two different edge nodes, which is uniformly distributed in the range of $[1, 10]$. Referring to the network price of Google [31], the unit cooperation cost $\tau_{n,m}$ is set to 0.004$/GB for each hop between node m and node n. The tasks are randomly generated at each edge server. We set the threshold for the overload probability constraint as 0.05. The chance constraint is always satisfied during the experiment. All results are averaged over 20 runs.

B. Benchmarks

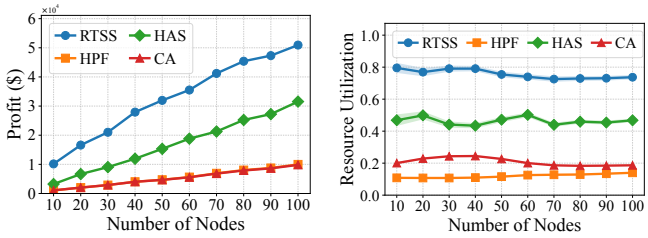
We compare the proposed RTSS algorithm with three typical benchmarks.

- **HPF**. Highest Profit First (HPF) is a classic method [32] to optimize task scheduling without considering overbooking. It sequentially iterates all possible task-server pairs and schedules these pairs in the descending order



(a) Profit (b) Resource utilization

Fig. 5. Performance with different numbers of tasks.



(a) Profit (b) Resource utilization

Fig. 6. Performance with different numbers of edge nodes.

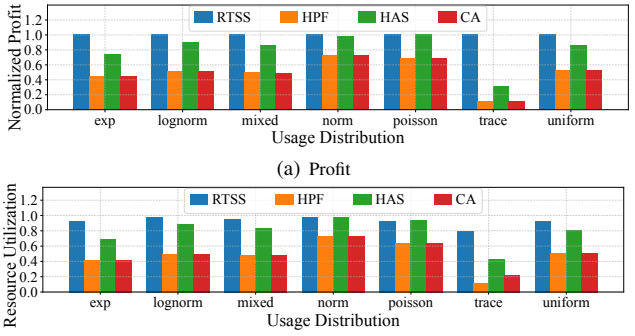
of their profit until the total requested computing resource reaches the computing resource capacity.

- **HAS** [22]. Hotspot-Aware Scheduling (HAS) algorithm is designed to schedule tasks in the cloud with overbooking. It formulates the problem into a probabilistic bin-packing problem and addresses it using the Γ-robustness theory. We compare against its offline version using recommended parameters (i.e., $p = 0.95$ and $t_p = 40$ [22]).
- **CA** [16]. The Contract Algorithm (CA) focuses on scheduling tasks on edge servers with overbooking. It investigates a series of matching games to establish stable contracts among the cloud, edge, and users. Tasks are then assigned according to the established contracts.

C. Results

Performance with different numbers of tasks. As shown in Fig. 5(a), benchmark methods (HPF, HAS, and CA) exhibit steady profit as the number of tasks increases. This occurs because these methods cannot effectively utilize the available resources even when the task pool is limited (e.g., 1000 tasks). In contrast, RTSS demonstrates a significant increase with growing task numbers, achieving average profit gains ranging from 1.06× to 2.47× over the best-performing baseline, HAS. Regarding resource utilization, as illustrated in Fig. 5(b), HPF, which does not consider overbooking, maintains relatively low resource utilization with an average value of only 11.0%. The matching-based approach (CA) improves resource utilization to 26.9%, while HAS achieves 47.2% through its overbooking capability. RTSS further enhances the resource utilization to 82.7%, which is 0.75× higher than HAS, demonstrating its superior ability to maximize resource efficiency.

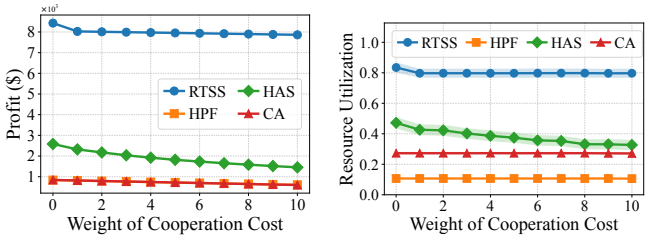
Performance with different numbers of edge nodes. As depicted in Fig. 6(a), all algorithms exhibit an overall increasing profit trend with the expansion of edge nodes. This improvement stems from the increased aggregate computing capacity, which enables the platform to accommodate a larger task workload. Across different node configurations, RTSS



(a) Profit

(b) Resource utilization

Fig. 7. Performance with different usage distributions.



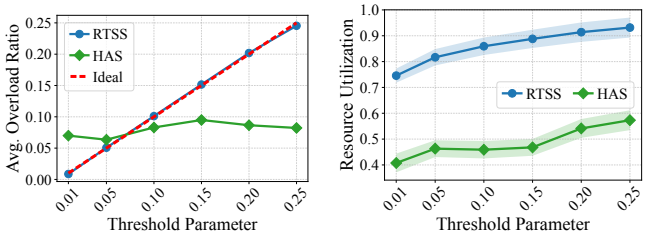
(a) Profit (b) Resource utilization

Fig. 8. Performance with different weights of cooperation cost.

achieves average profit gains of 5.71×, 1.13×, and 5.75× compared to HPF, HAS, and CA, respectively. Furthermore, Fig. 6(b) reveals that RTSS maintains superior resource utilization, delivering average improvements of 5.36×, 0.63×, and 2.66× over HPF, HAS, and CA, respectively. Notably, RTSS's resource utilization experiences a modest decline from 79.5% to 73.7% as the node count scales from 10 to 100, reflecting the diminishing task-to-resource ratio in the system.

Performance with different distributions of resource usage. We construct several synthetic distributions that the resource usage may follow [33], including exponential (exp), logarithmic normal (lognorm), normal (norm), Poisson (poisson), and uniform distributions (uniform), as well as a mixed distribution that uniformly samples from all types above. All synthetic distributions span $[0, 0.96]$ cores with requested resources of 0.96 cores. The exponential distribution has a mean of 0.40 cores, while others have a mean of 0.48 cores. We also include the real-world trace as a baseline reference. For comparison, profits are normalized by the maximum achieved under each distribution. As shown in Fig. 7, RTSS consistently outperforms other baselines across all distributions due to its distribution-agnostic design. Specifically, RTSS achieves average profit gains of 1.00×, 0.23×, and 1.01× and average resource utilization gains of 0.92×, 0.16×, and 0.86× compared to HPF, HAS, and CA, respectively. While HAS performs well under symmetric distributions, it fails on complex real-world traces that lack simple characteristics.

Performance with different cooperation costs. We investigate the impact of varying cooperation costs in Fig. 8, which represents low cost scenarios such as campus networks to high cost environments such as wide area networks. We multiply the cooperation cost $t_{r,n,m}$ by a weight factor ranging from 0 to 10. As the cooperation weight increases, HPF and CA maintain



(a) Average overload ratio (b) Resource utilization

Fig. 9. Performance with different overload threshold parameters.

relatively stable resource utilization around 10.6% and 27.2%, respectively, as they cannot effectively utilize the available computing resources. HAS, which incorporates overbooking, achieves an average resource utilization of 38.0%, but its profit and resource utilization decline as cooperation costs increase, as it is designed for cloud task scheduling and overlooks cooperation costs. RTSS, which is designed for edge task scheduling with overbooking, maintains a high average resource utilization of 80.0% and delivers average profit improvement of 10.21×, 3.35×, and 10.21× compared to HPF, HAS, and CA, respectively.

Accuracy of risk control. As algorithms designed for resource overbooking, both RTSS and HAS employ a threshold parameter (i.e., Δ used in Algorithm 2 and $1-p$ in HAS [22]) to constrain the probability of resource overload. We investigate how accurate these two algorithms can achieve in controlling the overload ratio (defined as the ratio of overloaded nodes to all nodes) corresponding to the threshold parameter. Fig. 9(a) reveals the key to RTSS's superiority in resource utilization: RTSS demonstrates remarkable precision in risk management by ensuring the observed ratio of overloaded nodes closely tracks the configured threshold. The discrepancy remains minimal, with a average deviation of 3.2%. Occasionally, the average overload ratio observed for RTSS slightly exceeds the specified threshold parameter. This minor deviation can be effectively mitigated in practical deployments by setting the threshold parameter marginally below the desired overload ratio. This figure also suggests that, thanks to RTSS's accurate risk control, system operators can tune the overload threshold to navigate the utilization–risk trade-off according to their requirements. In contrast, HAS adopts a more conservative strategy, resulting in a lower overloaded ratio but at the expense of reduced overall resource utilization. Consequently, RTSS delivers an average resource utilization improvement of 2.21× over HAS.

V. Related Works

Task scheduling without overbooking. Extensive research has been devoted to task scheduling, under the assumption that the actual resource consumption matches the requested allocation. For example, Liu et al. [34] use Lyapunov method to develop an efficient learning-based scheduling algorithm with deadline and throughput constraints. Liu et al. [35] propose a novel sunlight-aware heuristic algorithm to schedule tasks in the context of space edge computing. Luo et al. [29] develop a submodularity-based scheduling algorithm in geo-distributed clouds, achieving significant reductions in electricity costs. Zhao et al. [36] introduce a fast-convergence reinforcement learning algorithm to collaboratively schedule tasks. Sun et al. [37] propose a hybrid task scheduling algorithm to deal with online and offline tasks simultaneously. There are also some works that focus on jointly optimizing task scheduling with other decisions, such as resource allocation [38]–[40] and service placement [32], [41]–[46]. However, in real-world deployments, applications typically consume only a fraction of their reserved resources. The absence of overbooking considerations in these scheduling frameworks fundamentally constrains their practical applicability in production environments.

Task scheduling with overbooking. Research on overbooking-aware task scheduling is still in its infancy in both cloud and edge computing. In cloud computing, existing approaches typically model the problem as a stochastic bin-packing problem and utilize probability theory for solution development [21], [22], [47]. Recently, Wu et al. [22] proposed an efficient task scheduling algorithm based on Γ-robustness theory and protecting the physical machines from hotspots. However, they assume specific usage distribution features and overlook the cooperation cost between nodes, which limits their applicability in edge computing. In edge computing, several studies (e.g. [14]–[18]) explore trading contracts between cloud, edge, and users for contract-based scheduling. For instance, Liwang et al. [15] combine offline and online trading to handle the overbooking uncertainty, while Tang et al. [17] design auction mechanisms for dynamic resource overbooking to effectively maximize edge node profits. The fundamental difference between these works and RTSS is that they assume binary resource usage, while our approach handles arbitrary usage levels and better reflecting real-world scenarios. Tang et al. [23] leverage reinforcement learning for joint optimization of overbooking and task scheduling across edge nodes, but suffer from computational complexity and interpretability issues.

VI. Conclusion

In this paper, we investigate the resource overbooking problem in edge clouds, focusing on risk-aware task scheduling under uncertain resource usage. We formulate the problem as a stochastic mixed integer program and prove its NP-hardness. To address the challenge of quantifying and controlling overload risk without assuming specific usage distributions, we propose the Expected Utility-based Risk Evaluation (EURE) scheme, which enables precise risk assessment with additive error guarantees. Building on this, we design a submodular optimization-based scheduling algorithm that achieves a $(1-1/e)$-approximation ratio. Extensive experiments on real-world datasets demonstrate that our approach significantly improves both profit and resource utilization compared to state-of-the-art baselines, while maintaining accurate risk control. These results highlight the potential of risk-aware overbooking to enhance the efficiency and profitability of future edge computing platforms.

REFERENCES

[1] "Google distributed cloud," https://cloud.google.com/distributed-cloud, 2025.

[2] "Aws for the edge," https://aws.amazon.com/edge/, 2025.

[3] "Alibaba iot edge," https://www.alibabacloud.com/en/product/linkiotedge, 2025.

[4] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: workload analysis and scheduling in large-scale heterogeneous GPU clusters," in *USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2022, pp. 945–960.

[5] "Resource management for pods and containers," https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/, 2025.

[6] "Google cluster workload traces," https://www.kaggle.com/datasets/derrickmwiti/google-2019-cluster-sample, 2025.

[7] H. Yu, H. Wang, J. Li, X. Yuan, and S.-J. Park, "Accelerating serverless computing by harvesting idle resources," in *ACM Web Conf. (WWW)*, 2022, pp. 1741–1751.

[8] L. Tomás and J. Tordsson, "An autonomic approach to risk-aware data center overbooking," *IEEE Trans. Cloud Comput.*, vol. 2, no. 3, pp. 292–305, 2014.

[9] T. Jin, Z. Cai, B. Li, C. Zheng, G. Jiang, and J. Cheng, "Improving resource utilization by timely fine-grained scheduling," in *Eur. Conf. Comput. Syst. (EuroSys)*, 2020, pp. 1–16.

[10] X. Sun, C. Hu, R. Yang, P. Garraghan, T. Wo, J. Xu, J. Zhu, and C. Li, "ROSE: cluster resource scheduling via speculative over-subscription," in *IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2018, pp. 949–960.

[11] Z. Schwartz, T. D. Webb, M. Altin, and A. Riasi, "Overbooking and performance in hotel revenue management," *Int. J. Hosp. Manag.*, vol. 129, p. 104192, 2025.

[12] A. Nazifi, K. Gelbrich, Y. Grégoire, S. Koch, D. El-Manstrly, and J. Wirtz, "Proactive handling of flight overbooking: how to reduce negative ewom and the costs of bumping customers," *J. Serv. Res.*, vol. 24, no. 2, pp. 206–225, 2021.

[13] K. Chard and K. Bubendorfer, "High performance resource allocation strategies for computational economies," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 72–84, 2012.

[14] H. Qi, M. Liwang, S. Hosseinalipour, X. Xia, Z. Cheng, X. Wang, and Z. Jiao, "Matching-based hybrid service trading for task assignment over dynamic mobile crowdsensing networks," *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2597–2612, 2024.

[15] M. Liwang, Z. Gao, S. Hosseinalipour, Z. Cheng, X. Wang, and Z. Jiao, "Long-term or temporary? hybrid worker recruitment for mobile crowd sensing and computing," *IEEE Trans. Mobile Comput.*, vol. 24, no. 2, pp. 1055–1072, 2025.

[16] H. Qi, M. Liwang, X. Wang, L. Li, W. Gong, J. Jin, and Z. Jiao, "Bridge the present and future: a cross-layer matching game in dynamic cloud-aided mobile edge networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 12 522–12 539, 2024.

[17] Z. Tang, F. Zhang, X. Zhou, W. Jia, and W. Zhao, "Pricing model for dynamic resource overbooking in edge computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1970–1984, 2023.

[18] R. Chen, X. Wang, and X. Liu, "Smart futures based resource trading and coalition formation for real-time mobile data processing," *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 3047–3060, 2022.

[19] J. Kleinberg, Y. Rabani, and E. Tardos, "Allocating bandwidth for bursty connections," in *ACM Symp. Theory Comput. (STOC)*, 1997, pp. 664–673.

[20] M. C. Cohen, P. W. Keller, V. Mirrokni, and M. Zadimoghaddam, "Overcommitment in cloud services: bin packing with chance constraints," *Manag. Sci.*, vol. 65, no. 7, pp. 3255–3271, 2019.

[21] J. Yan, Y. Lu, L. Chen, S. Qin, Y. Fang, Q. Lin, T. Moscibroda, S. Rajmohan, and D. Zhang, "Solving the batch stochastic bin packing problem in cloud: a chance-constrained optimization approach," in *ACM SIGKDD Conf. Knowl. Discov. Data Min. (KDD)*, 2022, pp. 2169–2179.

[22] J. Wu, P. Popov, W. Yang, A. Gudkov, E. Ponomareva, X. Han, Y. Qiu, J. Song, and S. Romanov, "Hotspot-aware scheduling of virtual machines with overcommitment for ultimate utilization in cloud datacenters," *IEEE Trans. Autom. Sci. Eng.*, vol. 22, pp. 6809–6821, 2025.

[23] Z. Tang, F. Mou, J. Lou, W. Jia, Y. Wu, and W. Zhao, "Joint resource overbooking and container scheduling in edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 10 903–10 917, 2024.

[24] J. Li and A. Deshpande, "Maximizing expected utility for stochastic combinatorial optimization problems," in *IEEE Symp. Found. Comput. Sci. (FOCS)*, 2011, pp. 797–806.

[25] J. Li and A. Deshpande, "Maximizing expected utility for stochastic combinatorial optimization problems," *Math. Oper. Res.*, vol. 44, no. 1, pp. 354–375, 2019.

[26] P. R. Halmos, *Measure theory*. Springer, 2013.

[27] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[28] A. Krause and D. Golovin, "Submodular function maximization," *Tractability*, vol. 3, pp. 71–104, 2014.

[29] L. Luo, G. Zhao, H. Xu, Z. Yu, and L. Xie, "TanGo: a cost optimization framework for tenant task placement in geo-distributed clouds," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2023, pp. 1–10.

[30] "Google cloud price," https://cloud.google.com/compute/vm-instance-pricing, 2025.

[31] "Google network price," https://cloud.google.com/vpc/pricing, 2025.

[32] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 779–792, 2021.

[33] S. Liu, L. Pan, S. Liu, and K. Qi, "An online algorithm for inference service scheduling using combinations of server-based and serverless instances in cloud environments," *IEEE Internet Things J.*, vol. 12, no. 8, pp. 11 153–11 165, 2025.

[34] Q. Liu and Z. Fang, "Learning to schedule tasks with deadline and throughput constraints," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2023, pp. 1–10.

[35] W. Liu, Z. Lai, Q. Wu, H. Li, Q. Zhang, Z. Li, Y. Li, and J. Liu, "In-orbit processing or not? sunlight-aware task scheduling for energy-efficient space edge computing networks," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2024, pp. 881–890.

[36] Y. Zhao, C. Qiu, X. Shi, X. Wang, D. Niyato, and V. C. M. Leung, "Cur-coedge: curiosity-driven collaborative request scheduling in edge-cloud systems," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2024, pp. 411–420.

[37] Y. Sun, C. Lin, J. Ren, P. Wang, L. Wang, G. Wu, and Q. Zhang, "Subset selection for hybrid task scheduling with general cost constraints," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2022, pp. 790–799.

[38] Y. Li, T. Zeng, X. Zhang, J. Duan, and C. Wu, "Tapfinger: task placement and fine-grained resource allocation for edge machine learning," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2023.

[39] T. Ren, Z. Hu, H. He, J. Niu, and X. Liu, "FEAT: towards fast environment-adaptive task offloading and power allocation in MEC," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2023, pp. 1–10.

[40] Y. Liu, Y. Mao, Z. Liu, F. Ye, and Y. Yang, "Joint task offloading and resource allocation in heterogeneous edge environments," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2023, pp. 1–10.

[41] H. Wu, W. Lin, H. Zhang, F. Shi, W. Shen, K. Li, and A. Y. Zomaya, "Container scheduling strategy based on image layer reuse and sequential arrangement in mobile edge computing," *IEEE Trans. Mobile Comput.*, 2025, early Access.

[42] H. Liu, S. Liu, S. Long, Q. Deng, and Z. Li, "Joint optimization of model deployment for freshness-sensitive task assignment in edge intelligence," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2024, pp. 1751–1760.

[43] K. Peng, L. Wang, J. He, C. Cai, and M. Hu, "Joint optimization of service deployment and request routing for microservices in mobile edge computing," *IEEE Trans. Serv. Comput.*, vol. 17, no. 3, pp. 1016–1028, 2024.

[44] Z. Tang, J. Lou, and W. Jia, "Layer dependency-aware learning scheduling algorithms for containers in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 6, pp. 3444–3459, 2023.

[45] L. Wang, X. Liu, H. Ding, Y. Hu, K. Peng, and M. Hu, "Energy-delay-aware joint microservice deployment and request routing with dvfs in edge: a reinforcement learning approach," *IEEE Trans. Comput.*, vol. 74, no. 5, pp. 1589–1604, 2025.

[46] Y. Li, L. Gu, Z. Qu, L. Tian, and D. Zeng, "On efficient zygote container planning and task scheduling for edge native application acceleration," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2024, pp. 2259–2268.

[47] M. C. Cohen, P. W. Keller, V. Mirrokni, and M. Zadimoghaddam, "Overcommitment in cloud services: bin packing with chance constraints," *Manag. Sci.*, vol. 65, no. 7, pp. 3255–3271, 2019.