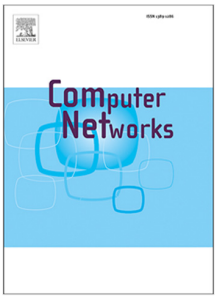
# Journal Pre-proof




Walking on two legs: Joint service placement and computation configuration for provisioning containerized services at edges

Tuo Cao, Qinhui Wang, Yuhan Zhang, Zhuzhong Qian, Yue Zeng, Mingtao Ji, Hesheng Sun, Baoliu Ye

PII: S1389-1286(23)00589-3
DOI: https://doi.org/10.1016/j.comnet.2023.110144
Reference: COMPNW 110144

To appear in: *Computer Networks*

Received date : 20 June 2023
Revised date : 15 November 2023
Accepted date : 15 December 2023

Please cite this article as: T. Cao, Q. Wang, Y. Zhang et al., Walking on two legs: Joint service placement and computation configuration for provisioning containerized services at edges, *Computer Networks* (2023), doi: https://doi.org/10.1016/j.comnet.2023.110144.

# Walking on Two Legs: Joint Service Placement and Computation Configuration for Provisioning Containerized Services at Edges

Tuo Cao, Qinhui Wang, Yuhan Zhang, Zhuzhong Qian, Yue Zeng, Mingtao Ji, Hesheng Sun, Baoliu Ye

**Abstract**

With the development of edge computing and container virtualization, provisioning containerized services at network edges is proposed for high responsiveness and low wide area network (WAN) traffic. However, realizing its full potential faces multiple challenges. First, due to containers' fine-grained computation resource isolation, finely configuring services' computation resource requirements is needed, especially for resource-constrained edge nodes. Second, due to edges' heterogeneity and services' diversity, system performance highly depends on which edge node to place each service. Third, as the main metric of quality of service, service response time involves the computing-network delay tradeoff and urges to optimize the decisions jointly. Prior works on edge-enabled service placement either ignore computation resource isolation and configuration, or assume computation resource configuration is given manually. To fill this gap, this paper investigates the joint service placement and computation configuration problem for provisioning containerized services at edges. Then based on the convex and submodular optimization techniques, we propose a two-stage greedy and local-search combined algorithm, TeLa for short. Rigorous theoretical analyses demonstrate that TeLa is a polynomial-time algorithm with performance guarantees. Finally, we implement twelve containerized services and an edge computing prototype to realistically evaluate TeLa. The results confirm TeLa's empirical superiority over state-of-the-art algorithms, in terms of 39% on average reduction on the weighted sum of service response time and WAN traffic.

*Keywords:* Containerized Service Provisioning, Edge Computing, Service Placement, Computation Configuration

## 1. Introduction

With the rapid development of container virtualization technologies, Container as a Service (CaaS) has emerged as a promising service model [1, 2, 3]. Unlike each virtual machine owning an entire operating system, containers share the system kernel with the host machine and encapsulate only the required libraries and tools. Thus, in addition to achieving runtime and resource isolation, containers acquire the advantages of fast launch, low overhead, easy deployment and migration, etc. Currently, representative CaaS platforms include Amazon Elastic Container Service [4], Google Kubernetes Engine [5] and Azure Container Instances [6]. Meanwhile, edge computing, which pushes the computing services from the cloud to the network edges, is proposed to mitigate the burden of cloud computing, i.e., long service response times and enormous wide area network (WAN) traffic [7, 8]. Evolution of the two prompts to implement services as containers and provision containerized services at network edges [9, 10, 11, 12].

Since managing containers at edges is still in its infancy [13] and Kubernetes (K8s) is the leading container orchestrator for cloud environments [14], practitioners usually adopt a K8s-like tool (e.g., MicroK8s [15], KubeEdge [16] and K3s [17]) to manage the edge-hosted containerized services. To be specific, they would first manually and empirically configure the amounts of resources that each containerized service exclusively occupies. The resource amounts, also called the resource requirements, are used for resource isolation among different services. Second, the K8s-like tool would place the containerized services onto the edge nodes, which is mainly according to services' resource requirements and nodes' resource capacities.

Such a containerized service provisioning scheme may work well for the cloud, but not for the edges. Firstly, edges' computation resources are scarce and containers' computation resource isolation is fine-grained, i.e., in processor percentages rather than numbers. They urge to *finely configure the computation resource requirements of containerized services (computation configuration)*, but the manual method is coarse-grained and wastes the resources. Fig. 1 is the analysis result obtained from Alibaba Group [18] and reveals the computation resource wastes. Secondly, edges are heterogeneous in resource capacities and access delays, and services are diverse in resource requirements and workloads. Besides, when edges are overloaded, the remaining services must stay at the cloud, bearing long

T. Cao, Y.H. Zhang, Z.Z Qian, Y. Zeng, M.T. Ji, H.S. Sun and B.L. Ye are with the State Key Laboratory for Novel Software Technology, the Department of Computer Science and Technology, Nanjing University, Nanjing Jiangsu 210023, China.

Q.H. Wang is with the Department of Military Training and Management, Army Command College, Nanjing Jiangsu 210045, China.

The corresponding author is Zhuzhong Qian (qzz@nju.edu.cn).

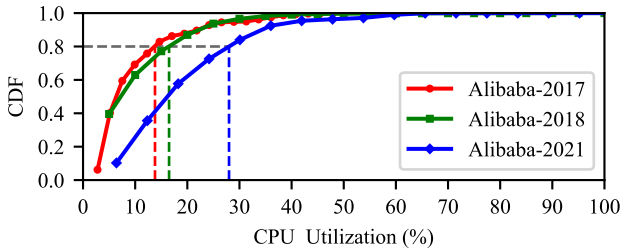



Fig. 1: CPU utilization of containerized services in Alibaba data centers in recent years [18], showing that more than 80% of its containerized services have average CPU utilization less than 30%.

network delays and incurring monetary expenses for WAN traffic [19, 20, 21]. It falls short to only consider resources when *deciding which node to place each containerized service (service placement)*. Lastly, computation configuration and service placement decisions are coupled but the two-step manner fails to capture it. For instance, configuring large computation resource requirements in the first step leads to low computing delays but may push services to stay at the cloud in the second step, increasing network delays. Hence, to make a computing-network delay tradeoff, the two decisions should be jointly optimized.

Overcoming these limitations raises the joint service placement and computation configuration problem, which is challenging to solve. On the one hand, service placement decisions are binary values, and computation configuration decisions are continuous values because of containers' fine-grained computation resource isolation. However, edges' limited computation resources and the computing-network delay tradeoff couple the two decisions deeply, in the sense that both their feasible sets and the performance metric (i.e., service response time) are hard to be separated with respect to them. It is challenging to decompose and resolve such problems [22]. On the other hand, with edges' heterogeneity and services' diversity considered, the service placement problem alone is hard to efficiently solve [23, 24], not to mention the joint optimization problem.

Existing works fall insufficient for handling these challenges. Some works focus on edge-enabled service placement or together with request scheduling [25, 26, 27, 28, 29, 30, 31, 32, 11, 12], but they neglect the service-level computation resource isolation and configuration. Some incorporate the isolation and configuration when placing services [23, 24, 33, 34, 35, 36, 37]. However, they assume the computation configuration is given manually or preset simply. A few works optimize service-level computation configuration [38, 39, 40, 41], but they assume each edge node hosts all services, or consider only one edge node.

To fill this gap, this paper investigates the compelling but less studied problem of joint service placement and computation configuration for provisioning containerized services at edges. Specifically, we aim at an edge computing system with multiple edge nodes and a remote cloud, which provides containerized services for users in a serving region. With containers' fine-grained computation resource isolation, edges' heterogeneity and services' diversity taken into account, we strive to minimize the weighted sum of service response time and WAN traffic, under multidimensional resource constraints. This problem is formulated as a mixed integer nonlinear program and proved to be NP-hard via the generalized assignment problem [42].

To handle the deep coupling relationship of the two decisions, we first apply an inner-and-outer problem model and decompose the joint problem into a computation configuration problem (inner) and a service placement problem (outer). Rigorous analyses show that such decomposition is nondestructive and lossless. We then prove the inner problem to be convex and obtain its closed-form solution through the Karush-Kuhn-Tucker conditions. After bringing the solution into the outer problem and making some arrangements, we prove the outer problem to be maximizing a submodular but non-monotone function over a $p$-independence system. To settle its NP-hardness, we raise a two-stage greedy and local-search combined algorithm, TeLa for short. It generates relatively good solutions via the greedy policy in the first stage and adjusts them via the local-search policy in the second stage. At last, we prove that TeLa is a $\frac{4/3-\epsilon}{p+2+1/p}$-approximation algorithm for the joint problem and runs in polynomial time.

Upon Java 11 and Docker [43], we implement twelve diverse containerized services and an edge computing prototype with four heterogeneous edges, as the testbed. We compare TeLa with state-of-the-art algorithms under various scenario settings. The results show that regarding the weighted sum of service response time and WAN traffic, TeLa outperforms the baselines by 39% on average and achieves near-optimal performance. Simulations in larger scales (more than 20 edges and 100 services) are also conducted and the results verify its scalability and efficiency.

The rest of this paper is organized as follows. In Section 2, we review the related works from two categories. Section 3 models the system and formulates the joint optimization problem. Section 4 presents the proposed algorithm TeLa and analyzes its performance. We evaluate TeLa through both testbed experiments and simulations in Section 5, and finally conclude this paper in Section 6.

## 2. Related Work

In recent years, provisioning services at network edges has been a hot topic and its service placement problem has drawn researchers' attention [44, 45, 46]. This section reviews the most related works from two categories and highlights the differences between their studies and ours.

### 2.1. *Edge-enabled Service Placement without Computation Configuration*

Works in this category optimize service placement at network edges while ensuring the actually used computation resources beneath the capacities. For example, Lia *et al.* [25] investigate this problem to minimize the exchanged

Table 1: Summary for Works on Edge-enabled Service Placement without Computation Configuration

| Ref. | Workload Type | Technique | Optimization Criterion | Evaluation Tool |
|---|---|---|---|---|
| [25] | General Services | Supervised Learning | Exchanged Intra-domain Traffic | Trace-driven Simulation |
| [26] | General Services | Gibbs Sampling | User Request Delays | Trace-driven Simulation |
| [27] | General Services | Lyapunov Optimization, Gibbs Sampling | Computation Latencies | Synthetic Simulation |
| [28] | General Services | Randomized Rounding | Number of Edge-served Requests | Synthetic Simulation |
| [29] | General Services | Submodular Optimization | Number of Edge-served Requests | Trace-driven Simulation |
| [30] | General Services | Submodular Optimization | Number of Edge-served Requests | Trace-driven Simulation |
| [31] | Live Streaming | Deep Reinforcement Learning | QoE Penalty, System Cost Penalty | Trace-driven Simulation |
| [32] | Live Streaming | Heuristics | Popularity-weighted Video Quality | Trace-driven Simulation |
| [11] | Container-based Microservices | Approximately Submodular Optimization | Number of Edge-served Requests | Trace-driven Simulation |
| [12] | Container-based Microservices | Heuristics | Service Latency Cost, Container Retention Cost | Trace-driven Simulation |

intra-domain traffic and explore the performance of several supervised learning techniques. Dou *et al.* [26] study the server placement and service placement problem to minimize the user request delay while Xu *et al.* [27] study the service placement and task offloading problem to minimize the computation latency. Besides, some works jointly optimize service placement and request scheduling to maximize the number of edge-served requests. In this direction, Poularakis *et al.* [28] propose a randomized rounding algorithm, He *et al.* [29] and Farhadi *et al.* [30] respectively design the greedy service placement (GSP) algorithm upon submodular optimization. In addition, some works focus on live streaming services and explore the video placement and viewer scheduling problem, where video streams may be real-timely transcoded by edge nodes. In this field, Wang *et al.* [31] raise a deep reinforcement learning algorithm to minimize the penalty of quality of experience (QoE) and system cost, while Lee *et al.* [32] raise a heuristic to maximize popularity-weighted video quality. Moreover, as the container virtualization technology develops, some researchers note containers' features and optimize containerized service provisioning at edges. For instance, Gu *et al.* [11] leverage the layered structure of container images and optimize service placement and request scheduling to maximize the number of edge-served requests. Noting containers' startup latencies, Pan *et al.* [12] raise a retention-aware service caching method to minimize service latency cost and container retention cost.

A comprehensive comparison of these works, in terms of the workload type, the technique, the optimization criterion and the evaluation tool, is summarized in Table 1. However, since they ignore the service-level computation resource isolation and configuration, services on the same node practically influence each other, which impairs the quality of service and even causes service crashes.

### *2.2. Edge-enabled Service Placement with Computation Configuration*

Works in this category optimize service placement at network edges, with the service-level computation resource isolation and configuration taken into account. From the perspective of algorithm design, Pasteris *et al.* [23] study this problem for a heterogeneous edge computing system and propose a heuristic to maximize the total system reward. Ouyang *et al.* [24] further observe the user mobility and the service migration cost, and propose an online algorithm to minimize the user-perceived latency within a service migration cost budget. Besides, Ma *et al.* [33] optimize service placement and workload scheduling to minimize service response time and outsourcing traffic, and develop an iterative caching update algorithm based on the idea of Gibbs sampling (GS) and water filling. Cao *et al.* [34] focus on the service placement and bandwidth allocation problem for edge-assisted mobile cloud gaming, and present an online algorithm to minimize the QoE impairment on network delays and frame rates. From the perspective of system design, Goethals *et al.* [35] present a Kubernetes-compatible edge container orchestrator to place containerized services on low-resource edge devices. Similar works include KubeEdge [16], K3s [17] and MicroK8s [15]. However, they all assume computation configuration is given manually by service providers, which usually wastes the computation resources. Additionally, Ouyang *et al.* [36] consider edge nodes' computation resources are evenly allocated to their hosted services, and optimize user-managed service placement to minimize user-perceived latencies and service switching cost. Li *et al.* [37] force each computing task to occupy one computing unit and raise an improved ant colony algorithm to place services to minimize the service response delay.

Note that there are a few works studying the service-level computation configuration problem for edge computing. For example, Xiang *et al.* [38] optimize computation configuration and traffic scheduling to minimize service response times within a computing expense budget, and Fan *et al.* [39] optimize computation configuration and request assignment to minimize service response times for Internet of things. Nevertheless, they assume each edge node hosts all services regardless of edges' memory and storage capacities, which is oversimplified and unrealistic. Moreover,

Table 2: Summary for Works on Edge-enabled Service Placement with Computation Configuration

| Ref. | Workload Type | Computation Configuration | Technique | Optimization Criterion | Evaluation Tool |
|---|---|---|---|---|---|
| [23] | General Services | Given Manually | Heuristics | System Reward | Synthetic Simulation |
| [24] | General Services | Given Manually | Lyapunov Optimization, Markov Approximation | User-perceived Latencies | Trace-driven Simulation |
| [33] | General Services | Given Manually | Gibbs Sampling, Convex Optimization | Response Times, Outsourcing Traffic | Synthetic Simulation |
| [34] | Mobile Cloud Gaming | Given Manually | Lyapunov Optimization, Markov Approximation | QoE on Network Delays, Frame Rates | Trace-driven Simulation |
| [35] | Container-based Services | Given Manually | OpenVPN, Containerd, Namespace, Cgroup | Resource Requirements | Testbed Experiment |
| [36] | General Services | Shared Evenly | Contextual Multi-armed Bandit, Thompson Sampling | Perceived Latencies, Switching Cost | Synthetic Simulation |
| [37] | General Services | One Unit for One Task | Ant Colony Optimization | Response Delays | Synthetic Simulation |
| [38] | General Services | Optimized | Lyapunov Optimization | Response Times | Synthetic Simulation |
| [39] | Internet of Things | Optimized | Convex Optimization | Response Times | Synthetic Simulation |
| [40] | Augmented Reality | Optimized | Successive Convex Approximation | Device Energy Consumption | Synthetic Simulation |
| [41] | General Services | Optimized | Dynamic Voltage and Frequency Scaling | Latencies, SLO Violation Ratios | Testbed Experiment |

Al-Shuwaili *et al.* [40] allocate computation resources to minimize device energy consumption for augmented reality applications, while Nam *et al.* [41] present EdgeIso, a light-weight scheduler that dynamically isolates tasks on edges, to minimize task latencies and service level objective (SLO) violation ratios. However, they consider only one edge node, instead of multiple heterogeneous edges.

In Table 2, we summarize a side-by-side comparison of these works, where the aspects include the workload type, the computation configuration method, the technique, the optimization criterion and the evaluation tool.

Different from all previous works, besides the edges' heterogeneity and the services' diversity, this paper further considers containers' fine-grained computation resource isolation, and investigates the joint service placement and computation configuration problem for provisioning containerized services at edges. To the best of our knowledge, this is the first work that specializes in this issue. Moreover, we propose a polynomial-time approximation algorithm for this joint problem and evaluate the algorithm through both testbed experiments and simulations.

## 3. System Model and Problem Formulation

In this section, we build the system model for provisioning containerized services at edges and formulate the joint service placement and computation configuration problem in detail. For ease of reference, we summarize the main notations used in this paper in Table 3.

### *3.1. System Overview*

As illustrated in Fig. 2, we consider an edge computing system that provides diverse computing services for the users or end devices in a specific serving region. To be precise, the system consists of i) a set $\mathcal{N}$ of heterogeneous edge nodes, deployed at the network edge and equipped with limited resources; and ii) a remote cloud $o$, composed of massive powerful servers and located far at the core network. They constitute the computing infrastructure for a system operator to deploy a set $\mathcal{S}$ of delay-sensitive but computing-intensive services, e.g., object detection, language translation, etc. In order to benefit from the container virtualization technology, the services are implemented as containers (namely, containerized services), instead of directly running on physical or virtual machines. Furthermore, each user or end device stochastically generates requests for some services, and transmits them to the edge node or the cloud that is hosting the corresponding service to get served. Finally, as for the serving region, it may be a manufacturing factory, a shopping mall, a railway station or other similar areas in practice.

Note that as time goes by, the system status, including service workloads, network delays, edge node availability, etc., may change stochastically or abruptly in reality. In order to adapt to the system dynamics, existing works either: i) adopt the slot-structured timeline and update their decisions every some time duration (e.g., a few minutes or more) [24, 30, 34]; or ii) adopt the event-triggering scheme and update their decisions when some predefined events (e.g., severe resource contentions) are detected [31, 41, 47]. Generally speaking, the time duration between two successive decision updates ranges from several minutes to hours, depending on the specific scenario. Following existing works [11, 25, 32, 37], this paper leaves the issue of determining when to update decisions to the system operator, and focuses on the service placement and computation configuration problem itself.

Table 3: Summary of Main Notations

| Symbols | Description |
|---|---|
| $\mathcal{N}/\mathcal{S}$ | Set of edge nodes/containerized services |
| $C_n/M_n$ | Computation/memory capacity of edge node $n$ |
| $G_n/B_n$ | Storage/bandwidth capacity of edge node $n$ |
| $m_s/g_s$ | Memory/storage requirement of service $s$ |
| $d_o/d_n$ | Network delay for communicating with remote cloud $o$/edge node $n$ |
| $\alpha_s$ | Expected computation resource demand (in CPU cycles) per request for service $s$ |
| $\beta_s$ | Expected data size per request for service $s$ |
| $\lambda_s$ | Expected arrival rate of requests for service $s$ |
| $c_o$ | Amount of computation resource that is allocated to a cloud-served request |
| $D_s/T_s$ | Service response time/WAN traffic per request for service $s$ † |
| $\omega$ | Weight parameter for balancing service response time and WAN traffic |
| **Decision** | **Description** |
| $x_{s,n}$ | Binary variables, indicating whether service $s$ is placed at node $n$ ($x_{s,n}=1$) or not ($x_{s,n}=0$) |
| $y_s$ | Continuous variables, specifying the computation resource amount configured for service $s$ (when placed on edges) to exclusively occupy |

† $D_s$ is controlled by service placement and computation configuration decisions. $T_s$ is controlled by service placement decisions.

### 3.2. Service Placement and Computation Configuration

Typically, the edge nodes together with the cloud are heterogeneous in resource capacities and access delays, and the services are diverse in resource requirements and workloads. As a consequence, the system performance heavily depends on where the services are placed and how many computation resources are configured for the services to exclusively occupy. We take the binary indicators $\boldsymbol{x} = (x_{s,n})_{s\in\mathcal{S},n\in\mathcal{N}\cup\{o\}}$ to denote the service placement decision. Definitely, let $x_{s,n} = 1$ if we place service $s \in \mathcal{S}$ on node $n \in \mathcal{N}\cup\{o\}$, and $x_{s,n} = 0$ otherwise. Here, node $n \in \mathcal{N}\cup\{o\}$ is short for edge node $n$ if $n \in \mathcal{N}$ or the cloud $o$ if $n = o$. As each service must be scheduled to one of the nodes to serve users, we have the following constraints:

$$\sum\nolimits_{n\in\mathcal{N}\cup\{o\}} x_{s,n} = 1, \quad \forall s \in \mathcal{S}, \tag{1}$$

$$x_{s,n} \in \{0,1\}, \forall s \in \mathcal{S}, \quad \forall n \in \mathcal{N}\cup\{o\}. \tag{2}$$

Unlike virtual machines isolating computation resources among services in the granularity of processor numbers, the container virtualization technology offers a much finer granularity, i.e., processor percentages. Intuitively, by taking advantage of this property, one could place more services on resource-limited edge nodes, improving their resource utilization and the system performance. Thus, without loss of generality, we take $\boldsymbol{y} = (y_s)_{s\in\mathcal{S}}$ to denote the computation configuration decision. Definitely, it means the amount of the computation resource exclusively occupied by service $s$ is $y_s$ CPU cycles per second. Note that in this paper, it is for presentation ease to use CPU cycles per second as the decision's unit and we can easily convert the unit into processor percentages. Therefore, we have

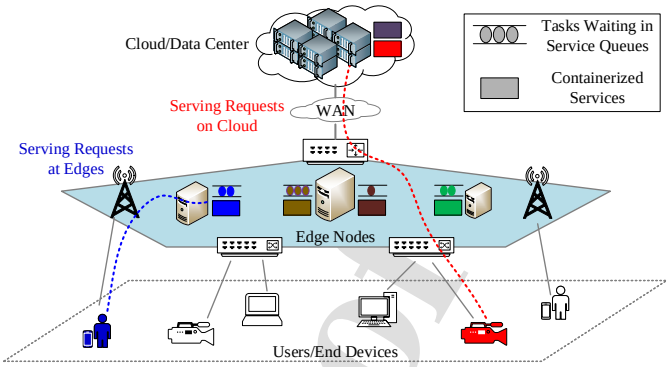



Fig. 2: Provisioning containerized services at edges.

$$y_s \geq 0, \quad \forall s \in \mathcal{S}. \tag{3}$$

Furthermore, to avoid the cold startup latency, containerized services are usually kept in memory with their exclusively occupied computation and storage resources. As a result, placing services on edge nodes inevitably consumes the resources of edge nodes, which are yet limited and scarce. For each edge node $n \in \mathcal{N}$, we take $C_n$, $M_n$ and $G_n$ to denote its computation, memory and storage capacity, respectively. For each service $s \in \mathcal{S}$, its memory and storage resource requirement could be obtained through several test runs and we take $m_s$ and $g_s$ to represent them. Since the occupied resources of an edge node should be upper bounded by the capacities, we have

$$\sum\nolimits_{s\in\mathcal{S}} x_{s,n} \times y_s \leq C_n, \quad \forall n \in \mathcal{N}, \tag{4}$$

$$\sum\nolimits_{s\in\mathcal{S}} x_{s,n} \times m_s \leq M_n, \quad \forall n \in \mathcal{N}, \tag{5}$$

$$\sum\nolimits_{s\in\mathcal{S}} x_{s,n} \times g_s \leq G_n, \quad \forall n \in \mathcal{N}. \tag{6}$$

### 3.3. Service Response Time and WAN Traffic

In general, the system operator would like to provision services for users with the best possible quality of service (QoS) and the minimum possible system cost. Specifically, for the QoS, a key and widely-used metric is the service response time, also known as the user-perceived delay or the request delay [26, 38, 39]. Meanwhile, when edge nodes are heavily loaded, some services have to be placed on the cloud and the user requests for cloud-hosted services have to be transmitted to the cloud across the WAN. On this occasion, the system operator has to pay for the WAN traffic, where the price is usually 0.01∼0.15 dollars per GB [19, 20, 21]. Thus, following [33, 36], this paper also treats the WAN traffic as the system cost metric, and proposes to minimize both the service response time and the WAN traffic. Furthermore, like prior works [27, 48], we consider that for any service $s \in \mathcal{S}$, the arrival of its user requests follows a Poisson process with expected rate $\lambda_s$ and the

computation demand per request (in CPU cycles) follows an exponential distribution with expectation $\alpha_s$. Besides, we take $\beta_s$ to denote the expected data size per request for service $s$, without restricting its probability distribution. In practice, the values of these parameters could be estimated or learned by analyzing the latest service log.

When it comes to modeling service response time and WAN traffic, placing services on edge nodes or the cloud is quite different. For an edge-hosted service $s$, the WAN traffic is zero because its user requests are all served at the network edge. The response time consists of network delays, queuing delays and computing delays. Since the locations of edge nodes may differ, we take $d_n$ to denote the average network delay for users to communicate with edge node $n$. Moreover, the edge serving process could be treated as an *M/M/1* model in the queuing theory [49] and the expected sojourn time (i.e., queuing time plus computing time) is calculated as $\frac{\alpha_s}{y_s-\alpha_s\lambda_s}$. Hence, the expected response time is $\sum_{n\in\mathcal{N}} x_{s,n}(d_n+\frac{\alpha_s}{y_s-\alpha_s\lambda_s})$. Stabilizing the waiting queues yields the following constraint:

$$y_s \geq \alpha_s\lambda_s, \quad \forall s\in\mathcal{S}, \sum\nolimits_{n\in\mathcal{N}} x_{s,n}=1 \tag{7}$$

Besides, taking $B_n$ to denote the bandwidth capacity of edge node $n$, we get the bandwidth resource constraint as

$$\sum\nolimits_{s\in\mathcal{S}} x_{s,n}\times\lambda_s\beta_s \leq B_n, \quad \forall n\in\mathcal{N}. \tag{8}$$

Whereas, for a cloud-hosted service $s$, its user requests are all transmitted to the cloud through the WAN and the WAN traffic per request is actually the expected data size, i.e., $\beta_s$. The network delay is mainly sourced from data propagation across the WAN and we take $d_o$ to denote it. Moreover, although the cloud has a large number of powerful servers, the resources for processing one user request are bounded, not infinite. Following existing works [50], we consider that once arriving at the cloud, any user request is immediately allocated a certain amount of resources for processing, where the computation resource amount is $c_o$. Therefore, the queuing delay is zero and the expected service response time is calculated as $x_{s,o}(d_o+\frac{\alpha_s}{c_o})$.

Hereafter, we take $D_s$ and $T_s$ to respectively denote the service response time and the WAN traffic per request for service $s\in\mathcal{S}$. Then, summarizing the two cases yields

$$D_s = x_{s,o}(d_o+\frac{\alpha_s}{c_o})+\sum_{n\in\mathcal{N}} x_{s,n}(d_n+\frac{\alpha_s}{y_s-\alpha_s\lambda_s}), \tag{9}$$

$$T_s = x_{s,o}\times\beta_s. \tag{10}$$

### *3.4. Problem Formulation*

With containers' fine-grained computation resource isolation, edges' heterogeneity and services' diversity taken into account, we jointly optimize service placement and computation configuration for provisioning containerized services at edges. The objective is to minimize the weighted sum of total service response time and total WAN traffic, subject to multidimensional resource constraints of edge nodes. Such a problem is formulated as follows:

$$\mathscr{P}_1: \min_{\boldsymbol{x},\boldsymbol{y}} \sum_{s\in\mathcal{S}}\lambda_s\times D_s+\omega\sum_{s\in\mathcal{S}}\lambda_s\times T_s$$

$$s.t. \sum\nolimits_{s\in\mathcal{S}} x_{s,n}\times y_s \leq C_n, \forall n\in\mathcal{N}, \tag{1a}$$

$$\sum\nolimits_{s\in\mathcal{S}} x_{s,n}\times m_s \leq M_n, \forall n\in\mathcal{N}, \tag{1b}$$

$$\sum\nolimits_{s\in\mathcal{S}} x_{s,n}\times g_s \leq G_n, \forall n\in\mathcal{N}, \tag{1c}$$

$$\sum\nolimits_{s\in\mathcal{S}} x_{s,n}\times\lambda_s\beta_s \leq B_n, \forall n\in\mathcal{N}, \tag{1d}$$

$$\sum\nolimits_{n\in\mathcal{N}\cup\{o\}} x_{s,n}=1, \forall s\in\mathcal{S}, \tag{1e}$$

$$(1-x_{s,o})y_s \geq (1-x_{s,o})\alpha_s\lambda_s, \forall s\in\mathcal{S}, \tag{1f}$$

$$x_{s,n}\in\{0,1\}, \forall s\in\mathcal{S}, n\in\mathcal{N}\cup\{o\}, \tag{1g}$$

$$y_s\geq 0, \forall s\in\mathcal{S}, \tag{1h}$$

where $\omega$ is the weight parameter to control the bias between service response time and WAN traffic. Constraints (1a)$\sim$(1d) are the computation, memory, storage and bandwidth resource constraints, respectively. Constraint (1e) ensures each service is placed on one edge node or the cloud and constraint (1f) is equivalent to Eq. (7), which stabilizes the waiting queues of edge-hosted services. Finally, constraints (1g) and (1h) specify the domains of service placement and computation configuration decisions.

Challenges to solving $\mathscr{P}_1$ are two-fold. First, with the presence of $D_s$, constraint (1a) and (1f), service placement decisions and computation configuration decisions are deeply coupled in the sense that the objective and the feasible set are both hard to be split with respect to them. Combining it with their value domains (i.e., binary and continuous, respectively) makes $\mathscr{P}_1$ a mixed integer nonlinear program. Second, the following theorem illustrates that the service placement problem alone is an NP-hard problem, not to mention the joint optimization problem.

**Theorem 1.** *The proposed problem $\mathscr{P}_1$ is NP-hard.*

*Proof.* Consider a simplified case of $\mathscr{P}_1$, where the edge nodes are assumed to have unlimited computation resources. Then the sojourn times of edge-hosted services in the objective function and the computation-related constraints (i.e., constraint (1a) and (1f)) could be neglected. Such a special problem is actually a multi-resource generalized assignment problem, which has been proven to be NP-hard [42]. Thus, as a general case, $\mathscr{P}_1$ is also NP-hard. □

*Remarks.* First, we emphasize the computation configuration decision $\boldsymbol{y}$, which specifies the computation resource amounts configured for edge-hosted services to exclusively occupy. According to Eq. (4) and (9), it greatly affects the service placement decision $\boldsymbol{x}$ and the service response time $D_s$. Meanwhile, it is upper bounded by the edge nodes' computation capacities (constraint (1a)) and lower bounded by the services' workloads (constraint (1f)). Second, the weight parameter $\omega$ is introduced to control

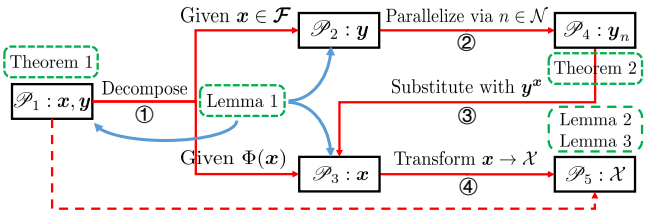



Fig. 3: Relationships of problems, theorems and lemmas. The blue lines indicate that Lemma 1 is related to $\mathscr{P}_1$, $\mathscr{P}_2$ and $\mathscr{P}_3$. The red dotted line indicates that $\mathscr{P}_1$ is eventually transformed into $\mathscr{P}_5$, indirectly through $\mathscr{P}_2$, $\mathscr{P}_3$ and $\mathscr{P}_4$.

the bias between the QoS (i.e., service response time) and the system cost (i.e., WAN traffic). Moreover, considering that the service response time per request generally ranges from tens of milliseconds to several seconds while the WAN traffic per request may range from bytes to megabytes, $\omega$ could also be used to normalize the two metrics to similar value scales. In practice, one could initially set $\omega$ to a large value (e.g., 1) to restrict the WAN traffic and the system cost, and gradually decrease $\omega$ to achieve the desirable balance between the QoS and the system cost.

## 4. Algorithm Design

To solve $\mathscr{P}_1$, this section develops a two-stage greedy and local-search combined algorithm, TeLa for short. In detail, as illustrated in Fig. 3, we first decompose $\mathscr{P}_1$ into a computation configuration subproblem $\mathscr{P}_2$ and a service placement subproblem $\mathscr{P}_3$. Then, we split $\mathscr{P}_2$ into subproblems $\mathscr{P}_4$ for parallelly solving, where each $\mathscr{P}_4$ corresponds to a unique edge node. Afterward, we solve $\mathscr{P}_4$ by the Karush-Kuhn-Tucker (KKT) conditions in convex optimization and bring its solutions into $\mathscr{P}_3$. Finally, we transform $\mathscr{P}_3$ into a set optimization problem $\mathscr{P}_5$ with desirable properties, and propose TeLa to solve $\mathscr{P}_5$ based on submodular optimization. In the end, we also provide the performance and complexity analysis of TeLa.

### 4.1. Problem Decomposition

As aforementioned, $\mathscr{P}_1$ is a mixed integer nonlinear program with binary decisions for service placement and continuous decisions for computation configuration. Since the two decisions belong to different problem families, i.e., combinatorial optimization and numerical optimization, a straightforward idea is to divide $\mathscr{P}_1$ into two independent subproblems and solve them separately. However, service response time and the computation resource constraint couple the two decisions deeply, implying that the derived subproblems should be solved simultaneously in an interactive manner. This subsection specifies how we decompose $\mathscr{P}_1$ to decouple the decisions and solving the subproblems is left for the next two subsections.

We start with the feasible set of service placement decisions, which we denote by $\boldsymbol{\mathcal{F}}$. To define it, the service-placement-related constraints, i.e., constraints (1a)$\sim$(1e) and (1g), are reserved. To eliminate the influence of computation configuration decisions on $\boldsymbol{\mathcal{F}}$, we relax constraint (1a) by leveraging constraints (1e) and (1f), and get

$$\boldsymbol{\mathcal{F}} = \{\boldsymbol{x} \mid (1b) \sim (1e), (1g), \sum\nolimits_{s\in\mathcal{S}} x_{s,n} \times \alpha_s\lambda_s \le C_n, \forall n \in \mathcal{N}\}.$$

To decompose $\mathscr{P}_1$, we then apply an inner-and-outer problem model, where solving the outer problem requires the solution to the inner problem. To be specific, for $\mathscr{P}_1$, the inner problem optimizes the computation configuration decision $\boldsymbol{y}$ under a given feasible service placement decision $\boldsymbol{x} \in \boldsymbol{\mathcal{F}}$, which is formulated as

$$\mathscr{P}_2 : \min_{\boldsymbol{y}} \sum_{s\in\mathcal{S}}\sum_{n\in\mathcal{N}} x_{s,n} \times (\lambda_s d_n + \frac{\alpha_s\lambda_s}{y_s - \alpha_s\lambda_s})$$
$$s.t.\ (1a), (1f).$$

For presentation ease, we denote its optimal solution under $\boldsymbol{x}$ by $\boldsymbol{y}^{\boldsymbol{x}}$ and its optimal value by $\Phi(\boldsymbol{x})$. It is worth noting that the definition of $\boldsymbol{\mathcal{F}}$ ensures the solvability of $\mathscr{P}_2$ and the existence of $\boldsymbol{y}^{\boldsymbol{x}}$. The outer problem treats computation configuration as an oracle and seeks the optimal service placement decision, which is formulated as

$$\mathscr{P}_3 : \min_{\boldsymbol{x}}\ \Phi(\boldsymbol{x}) + \sum_{s\in\mathcal{S}} x_{s,o} \times (\lambda_s d_o + \frac{\alpha_s\lambda_s}{c_o} + \omega\beta_s\lambda_s)$$
$$s.t.\ \boldsymbol{x} \in \boldsymbol{\mathcal{F}}.$$

**Lemma 1.** *Let $\boldsymbol{x}$ be a feasible solution to $\mathscr{P}_3$, then $\boldsymbol{x}$ together with $\boldsymbol{y}^{\boldsymbol{x}}$ is a feasible solution to $\mathscr{P}_1$. Moreover, $\mathscr{P}_3$ has the same optimal value with $\mathscr{P}_1$.*

*Proof.* See Appendix A for the detailed proof. □

*Remarks.* First, to decompose $\mathscr{P}_1$, one could alternatively let the inner problem optimize service placement and let the outer for computation configuration. However, in this way, the service placement problem is still NP-hard but the computation configuration problem turns to minimizing a discontinuous function, which is also intractable. Second, bringing $\mathscr{P}_2$'s objective into $\mathscr{P}_3$, it is easily observed that $\mathscr{P}_3$ has the same objective with $\mathscr{P}_1$. That is, the objective value of any $\boldsymbol{x} \in \boldsymbol{\mathcal{F}}$ and $\boldsymbol{y}^{\boldsymbol{x}}$ on $\mathscr{P}_3$ is the same as that on $\mathscr{P}_1$. Combining it with Lemma 1 implies that the decomposition method is nondestructive and lossless. In other words, an algorithm for $\mathscr{P}_3$ also solves $\mathscr{P}_1$ and its performance analysis for $\mathscr{P}_3$ applies to $\mathscr{P}_1$ as well.

### 4.2. Computation Configuration (Inner Problem)

Since solving the outer problem relies on the inner, we aim at the inner problem first. Observing $\mathscr{P}_2$, we learn that the computation configuration decision of one edge node has no influence on that of another. That is to say, each edge node could optimize its computation configuration independently and parallelly, which decreases the complexity without impairing the solution's optimality.

Therefore, we further split $\mathscr{P}_2$ into subproblems according to the edge nodes. Precisely, we take $\mathcal{S}_{\boldsymbol{x},n}$ to denote the set of services placed on edge node $n \in \mathcal{N}$ under service placement decision $\boldsymbol{x} \in \boldsymbol{\mathcal{F}}$, and take $\boldsymbol{y}_{\boldsymbol{x},n}$ to denote its computation configuration decision, i.e., $\mathcal{S}_{\boldsymbol{x},n} = \{s \in \mathcal{S} \mid x_{s,n} = 1\}$, $\boldsymbol{y}_{\boldsymbol{x},n} = (y_s)_{s \in \mathcal{S}_{\boldsymbol{x},n}}$. Then, for any $\boldsymbol{x} \in \boldsymbol{\mathcal{F}}$ and $n \in \mathcal{N}$, we propose to solve the following subproblem:

$$\mathscr{P}_4 : \min_{\boldsymbol{y}_{\boldsymbol{x},n}} \sum_{s \in \mathcal{S}_{\boldsymbol{x},n}} \frac{\alpha_s \lambda_s}{y_s - \alpha_s \lambda_s}$$

$$s.t. \sum\nolimits_{s \in \mathcal{S}_{\boldsymbol{x},n}} y_s \le C_n, \tag{4a}$$

$$y_s \ge \alpha_s \lambda_s, \forall s \in \mathcal{S}_{\boldsymbol{x},n}. \tag{4b}$$

**Theorem 2.** *$\mathscr{P}_4$ is a convex optimization problem.*

*Proof.* First, as the constraints of $\mathscr{P}_4$ are linear, the feasible set is convex. Second, we denote the objective function of $\mathscr{P}_4$ by $\Gamma(\boldsymbol{y}_{\boldsymbol{x},n})$ and for any services $s, s' \in \mathcal{S}_{\boldsymbol{x},n}$, we have

$$\frac{\partial^2 \Gamma}{\partial y_s \partial y_{s'}} = \begin{cases} = \dfrac{2\alpha_s \lambda_s}{(y_s - \alpha_s \lambda_s)^3} & s = s' \\ = 0 & s \ne s' \end{cases}.$$

Combining it with constraint (4b) makes the Hessian matrix $\boldsymbol{H} = (\frac{\partial^2 \Gamma}{\partial y_s \partial y_{s'}})_{|\mathcal{S}_{\boldsymbol{x},n}| \times |\mathcal{S}_{\boldsymbol{x},n}|}$ of $\Gamma(\boldsymbol{y}_{\boldsymbol{x},n})$ positive semi-definite. Therefore, $\Gamma(\boldsymbol{y}_{\boldsymbol{x},n})$ is a convex function over the feasible set. Summing them up, we conclude that $\mathscr{P}_4$ is a convex optimization problem. □

As $\mathscr{P}_4$ is proved to be convex, we apply the KKT conditions in convex optimization [51] to it and finally get the optimal computation configuration decision as follows:

$$y_s = \frac{C_n - \sum_{s' \in \mathcal{S}_{\boldsymbol{x},n}} \mu_{s'}}{\sum_{s' \in \mathcal{S}_{\boldsymbol{x},n}} \sqrt{\mu_{s'}}} \times \sqrt{\mu_s} + \mu_s, \forall s \in \mathcal{S}_{\boldsymbol{x},n},$$

where $\mu_s = \alpha_s \lambda_s$ is used for presentation brevity. Combining the decisions of all edge nodes, we have that given any $\boldsymbol{x} \in \boldsymbol{\mathcal{F}}$, the optimal solution to $\mathscr{P}_2$ is

$$y_s = \sum_{n \in \mathcal{N}} x_{s,n} \frac{C_n - \sum_{s' \in \mathcal{S}_{\boldsymbol{x},n}} \mu_{s'}}{\sum_{s' \in \mathcal{S}_{\boldsymbol{x},n}} \sqrt{\mu_{s'}}} \sqrt{\mu_s} + \mu_s, \forall s \in \mathcal{S}. \tag{11}$$

### *4.3. Service Placement (Outer Problem)*

Now, what remains is to solve the outer problem $\mathscr{P}_3$ for service placement. Since Theorem 1 has illustrated the NP-hardness of the service placement problem and it is impossible to optimally solve an NP-hard problem in polynomial time unless $P = NP$, we move on to design an efficient approximation algorithm with near-optimal performance. To begin with, we bring $\mathscr{P}_2$'s solution into $\mathscr{P}_3$ and rewrite $\mathscr{P}_3$'s objective function as follows:

$$\sum_{s \in \mathcal{S}} (x_{s,o}(\lambda_s d_o + \frac{\mu_s}{c_o} + \omega \beta_s \lambda_s) + \sum_{n \in \mathcal{N}} x_{s,n} \lambda_s d_n)$$
$$+ \sum_{n \in \mathcal{N}} \frac{(\sum_{s \in \mathcal{S}_{\boldsymbol{x},n}} \sqrt{\mu_s})^2}{C_n - (\sum_{s \in \mathcal{S}_{\boldsymbol{x},n}} \mu_s)}.$$

Observing the objective function of $\mathscr{P}_3$, we learn that the performance of placing any service on an edge node depends on not the service itself but the set of all services placed on that edge node. To leverage this set property, we transform $\mathscr{P}_3$ into a set optimization problem. Let $\mathcal{X} \subseteq \mathcal{S} \times \mathcal{N}$ denote the service placement decision, where $(s,n) \in \mathcal{X}$ means service $s \in \mathcal{S}$ is placed on edge node $n \in \mathcal{N}$. Given any decision $\mathcal{X}$, we denote the set of services that are placed on edge node $n$ by $\mathcal{S}_{\mathcal{X},n}$, i.e., $\mathcal{S}_{\mathcal{X},n} = \{s \in \mathcal{S} \mid (s,n) \in \mathcal{X}\}$. Then after making some arrangements, we get the set optimization version of $\mathscr{P}_3$ as

$$\mathscr{P}_5 : \max_{\mathcal{X}} \sum_{n \in \mathcal{N}} \sum_{s \in \mathcal{S}_{\mathcal{X},n}} (\lambda_s d_o + \frac{\mu_s}{c_o} + \omega \beta_s \lambda_s - \lambda_s d_n)$$
$$- \sum_{n \in \mathcal{N}} \frac{(\sum_{s \in \mathcal{S}_{\mathcal{X},n}} \sqrt{\mu_s})^2}{C_n - (\sum_{s \in \mathcal{S}_{\mathcal{X},n}} \mu_s)}$$

$$s.t. \sum\nolimits_{s \in \mathcal{S}_{\mathcal{X},n}} \mu_s \le C_n, \forall n \in \mathcal{N}, \tag{5a}$$

$$\sum\nolimits_{s \in \mathcal{S}_{\mathcal{X},n}} m_s \le M_n, \forall n \in \mathcal{N}, \tag{5b}$$

$$\sum\nolimits_{s \in \mathcal{S}_{\mathcal{X},n}} g_s \le G_n, \forall n \in \mathcal{N}, \tag{5c}$$

$$\sum\nolimits_{s \in \mathcal{S}_{\mathcal{X},n}} \lambda_s \beta_s \le B_n, \forall n \in \mathcal{N}, \tag{5d}$$

$$\mathcal{S}_{\mathcal{X},n} \cap \mathcal{S}_{\mathcal{X},n'} = \emptyset, \forall n, n' \in \mathcal{N}, n \ne n'. \tag{5e}$$

It is worth noting that $\mathscr{P}_5$ is a maximization problem, where the objective is actually the performance gain compared to placing all services on the cloud. For presentation brevity, we still take $\boldsymbol{\mathcal{F}} \subseteq 2^{\mathcal{S} \times \mathcal{N}}$ to denote the feasible set of $\mathscr{P}_5$ and it could be distinguished by the context. In addition, we take $\Omega(\mathcal{X}) : 2^{S \times N} \to \mathbb{R}$ to denote the objective function of $\mathscr{P}_5$. In what follows, we show that both $\Omega(\mathcal{X})$ and $\boldsymbol{\mathcal{F}}$ have desirable properties.

**Definition 1.** *([52]) A set function $f : 2^{\mathcal{U}} \to \mathbb{R}$ is monotone if $\forall \mathcal{U}_1 \subseteq \mathcal{U}_2 \subseteq \mathcal{U}$, $f(\mathcal{U}_1) \le f(\mathcal{U}_2)$ or $\forall \mathcal{U}_1 \subseteq \mathcal{U}_2 \subseteq \mathcal{U}$, $f(\mathcal{U}_1) \ge f(\mathcal{U}_2)$. Moreover, $f$ is submodular if $\forall \mathcal{U}_1 \subseteq \mathcal{U}_2 \subseteq \mathcal{U}$ and $u \in \mathcal{U} \backslash \mathcal{U}_2$, $f(\mathcal{U}_1 \cup \{u\}) - f(\mathcal{U}_1) \ge f(\mathcal{U}_2 \cup \{u\}) - f(\mathcal{U}_2)$.*

**Lemma 2.** *$\Omega(\mathcal{X})$ is submodular but non-monotone over $\boldsymbol{\mathcal{F}}$.*

*Proof.* See Appendix B for the detailed proof. □

**Definition 2.** *([53]) Let $\mathcal{U}$ be a universe of elements and $\mathcal{I}$ be a collection of subsets of $\mathcal{U}$, i.e., $\mathcal{I} \subseteq 2^{\mathcal{U}}$. $(\mathcal{U}, \mathcal{I})$ is called an independence system if: a) $\emptyset \in \mathcal{I}$; b) $\mathcal{U}_1 \subseteq \mathcal{U}_2$ and $\mathcal{U}_2 \in \mathcal{I}$ implies $\mathcal{U}_1 \in \mathcal{I}$. Then, the subsets in $\mathcal{I}$ are called independent and the inclusive-wise maximal independent set of $\mathcal{U}$ is called a basis of $\mathcal{U}$. Moreover, for a subset $\mathcal{T} \subseteq \mathcal{U}$, its rank $r(\mathcal{T})$ is defined as the cardinality of its largest basis and its lower rank $\rho(\mathcal{T})$ is the cardinality of its smallest basis. Then, an independence system $(\mathcal{U}, \mathcal{I})$ is called a p-independence system if $\max_{\mathcal{T} \subseteq \mathcal{U}} \frac{r(\mathcal{T})}{\rho(\mathcal{T})} \le p$.*

**Lemma 3.** *$(\mathcal{S} \times \mathcal{N}, \boldsymbol{\mathcal{F}})$ is a p-independence system, where $p = \min \{|\mathcal{N}|(\frac{\max_{n \in \mathcal{N}} C_n}{\min_{s \in \mathcal{S}} \mu_s}, \frac{\max_{n \in \mathcal{N}} M_n}{\min_{s \in \mathcal{S}} m_s}, \frac{\max_{n \in \mathcal{N}} G_n}{\min_{s \in \mathcal{S}} g_s}, \frac{\max_{n \in \mathcal{N}} B_n}{\min_{s \in \mathcal{S}} \lambda_s \beta_s}), |\mathcal{S}|\}$.*

**Algorithm 1:** Proposed Algorithm TeLa

**Input:** Problem parameters, i.e., $\mathcal{S}$, $\mathcal{N}$, $\lambda_s$, $\alpha_s$, $\beta_s$, $d_o$, $d_n$, etc., algorithm parameter $\epsilon$
**Output:** Decisions $\boldsymbol{x}$ and $\boldsymbol{y}$

// Initialization
1 $\mathcal{L} = \mathcal{S}, i = 0, \mathcal{X}^{(i)} = \emptyset, \mathcal{Y}^{(i)} = \emptyset$;
// Stage one: greedy exploration
2 **while** $\exists (s,n) \in \mathcal{L} \times \mathcal{N}, \mathcal{X}^{(i)} \cup \{(s,n)\} \in \boldsymbol{\mathcal{F}}$ **do**
3 $\quad (s^*, n^*) = \underset{\mathcal{X}^{(i)} \cup \{(s,n)\} \in \boldsymbol{\mathcal{F}}}{\arg\max} \Omega(\mathcal{X}^{(i)} \cup \{(s,n)\})$;
4 $\quad \mathcal{X}^{(i+1)} = \mathcal{X}^{(i)} \cup \{s^*, n^*\}$;
5 $\quad \mathcal{L} = \mathcal{L} \setminus \{s^*\}$;
6 $\quad i = i + 1$;
// Stage two: local-search exploitation
7 **for** $j = 1, 2, 3, \ldots, i$ **do**
8 $\quad (s^*, n^*) = \arg\max_{(s,n) \in \mathcal{X}^{(j)}} \Omega(\{(s,n)\})$;
9 $\quad \mathcal{Y}^{(j)} = \{(s^*, n^*)\}$;
10 $\quad$ **repeat**
11 $\quad\quad$ **if** $\exists (s', n') \in \mathcal{X}^{(j)} \setminus \mathcal{Y}^{(j)}, \Omega(\mathcal{Y}^{(j)} \cup \{(s', n')\}) > (1 + \frac{\epsilon}{|\mathcal{X}^{(j)}|})\Omega(\mathcal{Y}^{(j)})$ **then**
12 $\quad\quad\quad \mathcal{Y}^{(j)} = \mathcal{Y}^{(j)} \cup \{(s', n')\}$;
13 $\quad\quad$ **else if** $\exists (s', n') \in \mathcal{Y}^{(j)}, \Omega(\mathcal{Y}^{(j)} \setminus \{(s', n')\}) > (1 + \frac{\epsilon}{|\mathcal{X}^{(j)}|})\Omega(\mathcal{Y}^{(j)})$ **then**
14 $\quad\quad\quad \mathcal{Y}^{(j)} = \mathcal{Y}^{(j)} \setminus \{(s', n')\}$;
15 $\quad$ **until** $\mathcal{Y}^{(j)}$ *is not updated*;
16 $\quad$ **if** $\Omega(\mathcal{X}^{(j)} \setminus \mathcal{Y}^{(j)}) > \Omega(\mathcal{Y}^{(j)})$ **then**
17 $\quad\quad \mathcal{Y}^{(j)} = \mathcal{X}^{(j)} \setminus \mathcal{Y}^{(j)}$ ;
18 $\mathcal{X}^* = \arg\max_{\mathcal{X} \in \{\mathcal{X}^{(j)}\}_{j=0}^{i} \cup \{\mathcal{Y}^{(j)}\}_{j=0}^{i}} \Omega(\mathcal{X})$;
19 Obtain $\boldsymbol{x}$ from $\mathcal{X}^*$ and obtain $\boldsymbol{y}$ by Eq. (11);
20 **return** $\boldsymbol{x}$ and $\boldsymbol{y}$.

*Proof.* See Appendix C for the detailed proof. □

Since $\mathscr{P}_5$ is proved to be maximizing a submodular but non-monotone function over a $p$-independence system, we propose a two-stage greedy and local-search combined algorithm TeLa based on the framework in [53]. In detail, TeLa works as follows (also shown in Algorithm 1). Initially, auxiliary decisions $\mathcal{X}^{(0)}$ and $\mathcal{Y}^{(0)}$ are set to empty (Line 1), which means all services are placed on the cloud. Here, $\mathcal{L}$ is the set of services that still stay on the cloud and $i$ is the iteration index. Following is the first stage, which applies the greedy policy (Lines 2-6). TeLa puts the pair $(s,n)$ that has the largest performance margin gain into auxiliary decisions in each iteration until no service could be placed on any edge node. Since the objective function $\Omega(\mathcal{X})$ is non-monotone, its value may decrease as this process evolves. Hence, we develop the second stage, which applies the local-search policy to seek as better solutions as possible (Lines 7-17). Specifically, for decision $\mathcal{X}^{(j)}$ of the $j$-th iteration in the first stage, TeLa iteratively adopts local neighbors that improve the performance by more than $\frac{\epsilon}{|\mathcal{X}^{(j)}|}$ and gives the best decision $\mathcal{Y}^{(j)}$ ever encountered. At last, TeLa chooses the best solution that is found in the whole process (Line 18) and converts it into the original decision forms, i.e., $\boldsymbol{x}$ and $\boldsymbol{y}$ (Line 19).

*Remarks.* First, by Lines 3 and 4, TeLa ensures each $\mathcal{X}^{(j)}$ is a feasible solution to $\mathscr{P}_5$. Meanwhile, the second stage of TeLa ensures each $\mathcal{Y}^{(j)}$ is a subset of $\mathcal{X}^{(j)}$. Then according to Lemma 3, each $\mathcal{Y}^{(j)}$ is also feasible for $\mathscr{P}_5$, though they are not explicitly checked. Second, we emphasize the algorithmic parameter $\epsilon$ in Lines 11 and 13. It is introduced to balance the performance and the complexity of TeLa. In general, a smaller $\epsilon$ drives TeLa to spend more computing time to achieve better performance.

### *4.4. Performance Analysis*

**Theorem 3.** *TeLa is a $\frac{4/3-\epsilon}{p+2+1/p}$-approximation algorithm for $\mathscr{P}_5$, where $p$ is defined in Lemma 3 and $\epsilon$ is a small positive algorithmic parameter. Its computation complexity is $\mathcal{O}(|\mathcal{S}|^3|\mathcal{N}| + |\mathcal{S}|^2|\mathcal{N}|^2 + \frac{1}{\epsilon}|\mathcal{S}|^4 \lg|\mathcal{S}| + \frac{1}{\epsilon}|\mathcal{S}|^3|\mathcal{N}| \lg|\mathcal{S}|)$.*

*Proof.* Combining Lemma 2 and Lemma 3, we have that $\mathscr{P}_5$ is maximizing a non-monotone submodular function over a $p$-independence system. Then, according to Theorem 2 in [53] and Theorem 3.4 in [54], TeLa is a $\frac{4/3-\epsilon}{p+2+1/p}$-approximation algorithm for $\mathscr{P}_5$. Note that $\mathscr{P}_5$ optimizes service placement and computation configuration to maximize the performance gain compared to placing all services on the cloud, where the performance is the weighted sum of total service response time and total WAN traffic.

The computation complexity of TeLa mainly depends on the two stages. For stage one, as each iteration would offload one service from the cloud to the edge nodes, there are at most $|\mathcal{S}|$ iterations. In each iteration, TeLa computes $\Omega$ for at most $|\mathcal{S}| \times |\mathcal{N}|$ times. The computation complexity for computing $\Omega$ is $\mathcal{O}(|\mathcal{S}| + |\mathcal{N}|)$. Hence, the computation complexity of stage one is $\mathcal{O}(|\mathcal{S}|^3|\mathcal{N}| + |\mathcal{S}|^2|\mathcal{N}|^2)$. For stage two, the number of $\mathcal{Y}^{(j)}$s is up to $|\mathcal{S}|$. According to [54], computing each $\mathcal{Y}^{(j)}$ involves computing $\Omega$ for at most $\mathcal{O}(\frac{1}{\epsilon}|\mathcal{S}|^2 \lg|\mathcal{S}|)$ times. Thus, the computation complexity of stage two is $\mathcal{O}(\frac{1}{\epsilon}|\mathcal{S}|^4 \lg|\mathcal{S}| + \frac{1}{\epsilon}|\mathcal{S}|^3|\mathcal{N}| \lg|\mathcal{S}|)$. Summing them up, we get TeLa's computation complexity as $\mathcal{O}(|\mathcal{S}|^3|\mathcal{N}| + |\mathcal{S}|^2|\mathcal{N}|^2 + \frac{1}{\epsilon}|\mathcal{S}|^4 \lg|\mathcal{S}| + \frac{1}{\epsilon}|\mathcal{S}|^3|\mathcal{N}| \lg|\mathcal{S}|)$. □

## 5. Implementation and Evaluation

In this section, we implement twelve containerized services and an edge computing prototype to evaluate TeLa. We compare it with state-of-the-art algorithms under various scenario settings. Simulations in larger scales are also conducted to verify its scalability and time efficiency.

### *5.1. Testbed Implementation*

**Implementation Details.** We first implement twelve containerized services with their client programs, covering language translation, speech recognition, document conversion, face recognition, word counting, photo enhancement and blind watermarking. By running various tasks of

Table 4: Summary of Containerized Services

| Index | $m_s$ (MB) | $g_s$ (MB) | $\beta_s$ (KB) | $\alpha_s$ † | $\lambda_s$ † |
|---|---|---|---|---|---|
| 1 | 800 | 117 | 20.5 | 3.08 | 1 |
| 2 | 3500 | 3350 | 2.27 | 84.1 | 0.1 |
| 3 | 800 | 1830 | 433 | 33.2 | 0.1 |
| 4 | 800 | 916 | 208 | 19.4 | 0.5 |
| 5 | 800 | 916 | 1690 | 14.8 | 0.5 |
| 6 | 100 | 85 | 14.1 | 0.123 | 3 |
| 7 | 4500 | 2800 | 1570 | 144 | 0.1 |
| 8 | 500 | 1120 | 1910 | 15.0 | 0.2 |
| 9 | 500 | 1120 | 300 | 12.7 | 0.5 |
| 10 | 2500 | 711 | 894 | 7.42 | 0.5 |
| 11 | 1000 | 732 | 619 | 5.23 | 0.2 |
| 12 | 1000 | 702 | 2360 | 3.57 | 1 |

† The units of $\alpha_s$ and $\lambda_s$ are giga CPU cycles per request and requests per second, respectively.

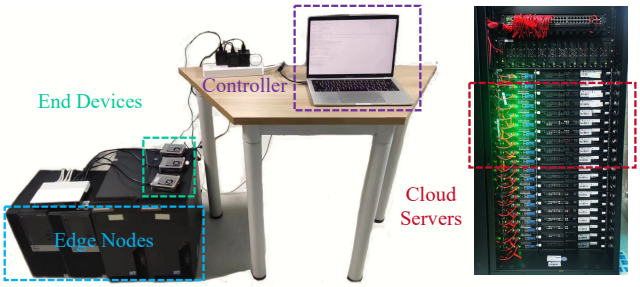



Fig. 4: Infrastructure of our testbed system.

each service on one node for more than one hundred times, we learn that they have memory requirements $m_s$ ranging from 100 MB to 4500 MB, storage requirements $g_s$ from 85 MB to 3350 MB, average request data sizes $\beta_s$ from 2.27 KB to 2.36 MB, and expected computation demands per request $\alpha_s$ from 0.123 to 144 giga CPU cycles. Moreover, we set the request arrival pattern of each service to follow a Poisson process, where the expected rate $\lambda_s$ ranges from 0.1 to 3 requests per second. The detailed information of these containerized services is listed in Table 4.

To deploy the containerized services, we build an edge computing system with four heterogeneous desktops as edge nodes, several servers as the cloud and a laptop as the controller, as shown in Fig. 4. The edge nodes have the memory capacities $M_n \in \{4, 8, 16\}$ GB and the storage capacities $G_n \in \{512, 1024\}$ GB. The computation capacity $C_n$ is set to the product of the processor number and the CPU frequency, ranging from 11.2 GHz to 19.2 GHz, and the bandwidth capacity $B_n$ is 1 Gbps. Furthermore, each cloud server has 32 CPUs with the frequency of 2.1 GHz and we allocate 2 CPUs to any cloud-executed task. Thus, the computation resource that is available to one cloud-executed task, i.e., $c_o$, is 4.2 GHz. Following existing works [55, 36, 56], the network delay for the cloud $o$, i.e., $d_o$, is set to 100 ms and that for any edge node $n \in \mathcal{N}$, i.e., $d_n$, is distributed in [5, 15] ms. We summarize their specifications in Table 5. Besides, three Raspberry Pis

Table 5: Specifications of Edge Nodes and Cloud Servers

| ID | CPU | Mem. | Sto. | Ban. | Del. |
|---|---|---|---|---|---|
| EN1 | 12.8 GHz | 8 GB | 1 TB | 1 Gbps | 5∼15 ms |
| EN2 | 11.2 GHz | 4 GB | 512 GB | 1 Gbps | 5∼15 ms |
| EN3 | 11.2 GHz | 4 GB | 512 GB | 1 Gbps | 5∼15 ms |
| EN4 | 19.2 GHz | 16 GB | 1 TB | 1 Gbps | 5∼15 ms |
| Clo.† | 67.2 GHz | 64 GB | 1 TB | 1 Gbps | 100 ms |

† The specifications of one cloud server.

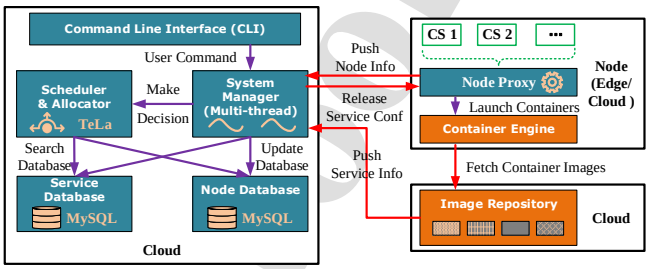



Fig. 5: Framework of our testbed system.

are used to act as end devices to run the client programs.

Then we implement our testbed system using Java 11 and Docker [43] with its image repository. As illustrated in Fig. 5, the system consists of six modules. The databases maintain the information of containerized services and edge nodes, respectively. The node proxy is for communicating with the system manager and launching containerized services. The scheduler and allocator invokes TeLa to optimize service placement and computation configuration decisions, where the inputs are obtained from the databases and the outputs are returned to the system manager. Here, the parameter $\epsilon$ is set to $10^{-2}$. Based on these modules, the system manager controls the system with the multi-thread mechanism for high responsiveness. Lastly, the CLI module is used for interacting with the system operator.

**Benchmarks.** As shown in Table 1 and Table 2, some existing works solve the service placement problem via Gibbs sampling, which is an optimization technique for general combinatorial problems [26, 27, 33]. Some works note the problem's submodularity and propose the greedy service placement algorithm [29, 30, 11]. Moreover, many works consider the computation configuration decisions are given manually in the granularity of processor numbers [23, 24, 33, 34]. Intuitively, configuring smaller computation resources makes one to place more services at edges to achieve shorter network delays and lower WAN traffic. Thus, we compare TeLa with the following algorithms:

- **GSP-C** is a variant of the greedy service placement algorithm [29] and makes the two decisions simultaneously. For service placement, it iteratively selects the pair $(s, n)$ that has the highest marginal performance gain and places service $s$ on edge node $n$. Its computation configuration decision is coarse-grained, and is set to the minimum processor number that meets the computation resource constraints.

Table 6: Number of Cloud-hosted Services under Various Weights

| Alg. \ Wei. | $10^{-2}$ | $10^{-3}$ | $10^{-4}$ | $10^{-5}$ | $10^{-6}$ | 0 |
|---|---|---|---|---|---|---|
| GS-C | 5 | 5 | 5 | 8 | 10 | 11 |
| GSP-C | 5 | 5 | 5 | 9 | 9 | 11 |
| TeLa | 3 | 3 | 3 | 7 | 7 | 7 |
| Optimal | 2 | 2 | 3 | 7 | 7 | 8 |

- **GS-C** is a variant of Gibbs sampling [33] and shares the same computation configuration method with GSP-C. For service placement, it uses a randomized local-search-based policy. In each iteration, it randomly selects one of the neighbors and adopts it with a performance-related probability. We set its parameter to $10^{-4}$ (small enough) and stop it when the decision has been unchanged for 10 iterations.
- **Optimal** uses our computation configuration method and makes the service placement decision by exhaustive search. It has the exponential computation complexity and takes a long time to complete.

For every experiment setting, we run the system for over 15 minutes, during which more than 7000 user requests are received and processed. Since the system runs may differ in total request numbers, we mainly compare these algorithms in terms of response time per request, WAN traffic per request and the weighted sum of response time and WAN traffic per request. Unless specified otherwise, the experiment settings are the same as above.

### 5.2. Testbed Evaluation Results

Since the weight parameter $\omega$, which controls the bias between service response time and WAN traffic, affects the decision-making process and the system performance, we first conduct preliminary experiments to decide its value. We initially set $\omega$ to 1 and gradually reduce it to 0, and Table 6 shows the main results regarding the number of cloud-hosted services. On the one hand, when $\omega$ is greater than $10^{-4}$, as less services as possible are placed at the cloud to reduce WAN traffic and the edges are heavily loaded. In this case, the cloud-hosted service number turns constant and it seems the algorithms are irrespective of $\omega$. On the other hand, when $\omega$ is less than $10^{-5}$, service response time becomes much more important than WAN traffic, and the algorithms' decisions tend to be relatively stable, especially for TeLa and Optimal. In conclusion, in order to involve and evaluate the impact of $\omega$, we would better make its value in the range of $[10^{-4}, 10^{-5}]$. Hence, in the following experiments, we either set $\omega$ to $5 \times 10^{-5}$ by default, or vary its value from $10^{-4}$ to $10^{-5}$.

Then we conduct experiments to compare the algorithms' performance under various weight parameters, and illustrate the results in Fig. 6. As expected, a larger value of $\omega$ drives the algorithms to focus more on WAN traffic

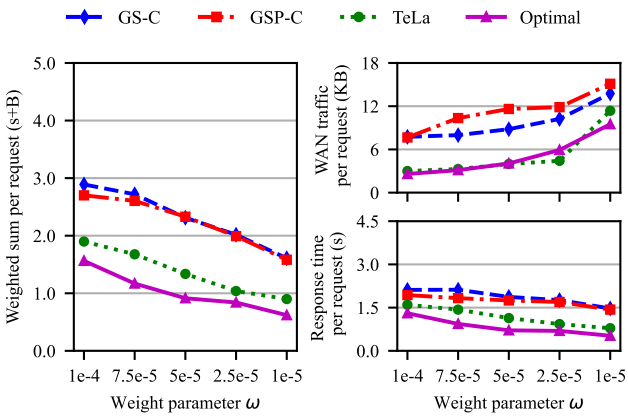
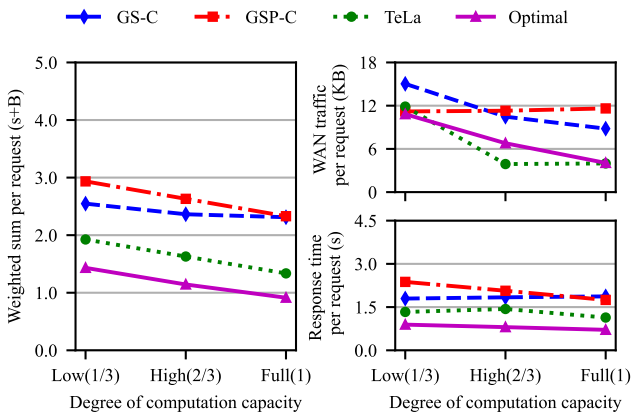


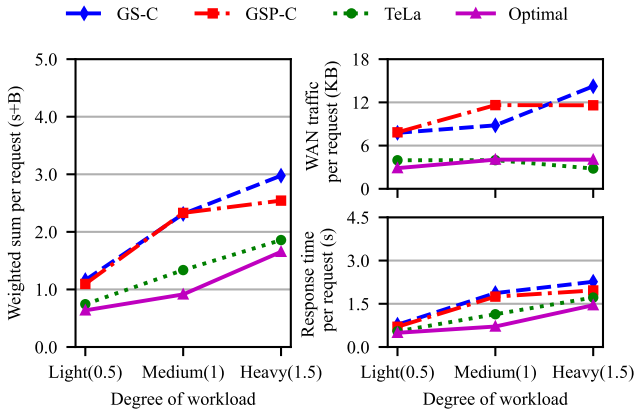

Fig. 6: Testbed results under various weight parameters.

and less on response time, and vice versa. Nevertheless, TeLa and Optimal always perform better than GS-C and GSP-C, with a 29.7% to 61.5% reduction on the weighted sum per request. The reasons are two-fold. First, thanks to the fine-grained computation configuration, TeLa and Optimal can make full use of edges' resources and place fewer services (i.e., 3 to 7) on the cloud. By contrast, GS-C and GSP-C have to place more services (i.e., 5 to 9) on the cloud and generate more WAN traffic. Second, due to the application of convex optimization, TeLa and Optimal can make the optimal computation configuration decision to minimize response time. Whereas, adopting a heuristic and coarse-grained computation configuration method, GS-C and GSP-C utilize the computation resources inefficiently and bear longer response time. Note that though TeLa and Optimal sometimes have the same cloud-hosted service number, they may place services onto different nodes and perform differently in response time, as a result of edges' heterogeneity and services' diversity.

Next, in Fig. 7(a) and Fig. 7(b), we measure the impact of edges' computation capacities and services' workloads on the algorithms' performance. Here, low, high and full capacities refer to setting edges' computation capacities to $\frac{C_n}{3}$, $\frac{2C_n}{3}$ and $C_n$, and light, medium and heavy workloads refer to setting the expected request arrival rates to $0.5\lambda_s$, $\lambda_s$ and $1.5\lambda_s$. Clearly, increasing edges' capacities or decreasing services' workloads result in lower response time and WAN traffic, since edges become relatively more capable. However, the algorithms differ in the adaptation behaviors. For instance, when edges' computation capacities advance from low to full, the cloud-hosted service number of TeLa and Optimal changes from 7 to 5, that of GS-C changes from 8 to 6, while GSP-C still places 7 services on the cloud and only adapts service placement among the heterogeneous edges. When services' workloads decrease from heavy to light, GS-C and GSP-C offload more services to edges while TeLa and Optimal mainly adapt computation configuration and service placement among the edges. In summary, covering such variations, TeLa reduces the weighted sum per request of GS-C and GSP-C by 24.4% to 42.2% and 26.9% to 42.6%, respectively.

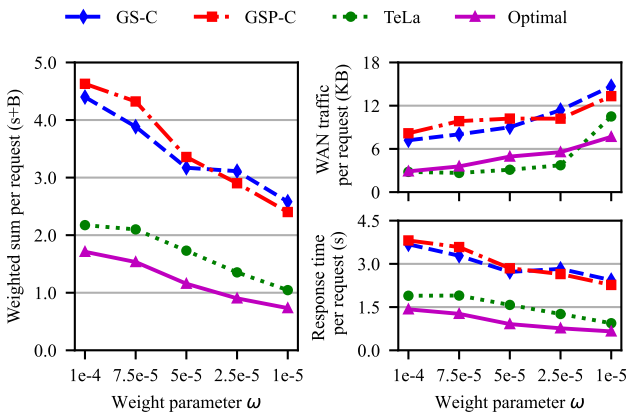



(a) Impact of edges' computation capacities

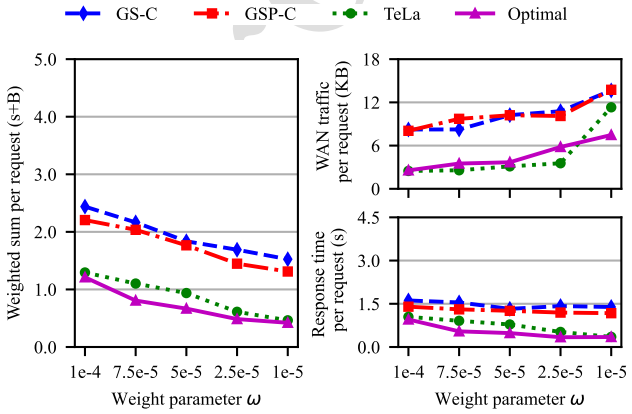



(b) Impact of services' workloads

Fig. 7: Testbed results under various edge node computation capacities and service workloads.




(a) Impact of request arrival patterns (burst pattern)




(b) Impact of request arrival patterns (uniform pattern)

Fig. 8: Testbed results under various request arrival patterns.

To further evaluate the robustness of TeLa, we change the request arrival patterns from Poisson to burst and uniform, and visualize their effects in Fig. 8(a) and Fig. 8(b), respectively. In the burst pattern, users send service requests every ten seconds, leading to a series of workload bursts, while in the uniform pattern, the time intervals of user requests are uniformly distributed in $[\frac{0.5}{\lambda_s}, \frac{1.5}{\lambda_s}]$ seconds. Due to the unawareness of the request arrival pattern variations, these algorithms do not adapt their decisions. Thus, their cloud-hosted services remain the same and their performances on WAN traffic hardly change. However, concerning the response time, their performances change notably. In detail, the burst pattern makes more requests to wait in queue and bear longer queuing delays, driving the algorithms to perform poorer. By contrast, the uniform pattern makes request arrivals steady and decreases queuing delays. Nonetheless, TeLa reduces the response time per request of GS-C and GSP-C by 35.2% to 74.8% and 25.2% to 70.2%. It implies that thanks to the optimized and fine-grained computation configuration, TeLa could utilize the resources more effectively.

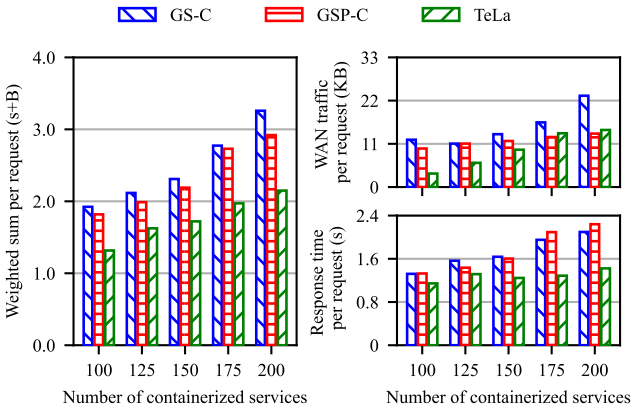
### 5.3. Simulation Evaluation Results

To further evaluate TeLa's scalability, simulations in larger scales are also conducted. Specifically, the service and edge node numbers are by default 150 and 30, and their specifications (e.g., resource capacities and resource requirements) are uniformly sampled from our implemented services and edge nodes. We then vary the two numbers from 100 to 200 and 20 to 40, and show their impact on the algorithms in Fig. 9. Due to the fact that Optimal has the exponential computation complexity and spends an unacceptably long time to compute, e.g., more than an hour for even 10 edges and 30 services, we omit its performance here and focus on the rest algorithms. Since the edge nodes are resource-limited, a larger service number means that the edges are more likely to be heavily loaded and more services would be placed on the cloud, leading to longer response time and larger WAN traffic. In our simulations, the average cloud-hosted service ratio under 100 services is 48.7% and that under 200 services is 67.7%. On the contrary, a larger edge node number leads to shorter response time and smaller WAN traffic, since more resources are deployed near the users. Covering these settings, TeLa achieves a 15.8% to 42.2% reduction on the weighted sum per request, implying that TeLa scales well.

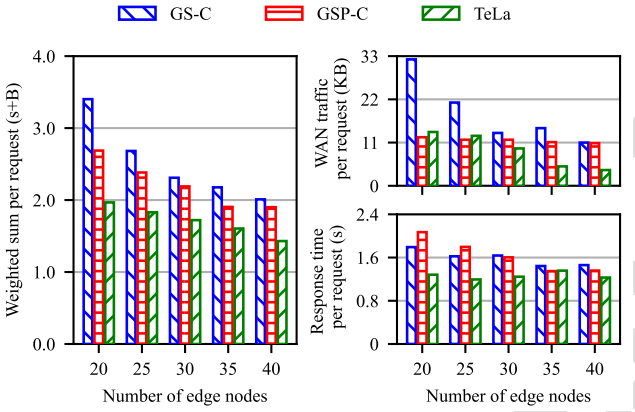
Finally, to evaluate the time efficiency of TeLa, we run these algorithms on an Intel Core I7-7800X@3.5GHz CPU, and show their computing times for 50∼300 services and 25∼150 edge nodes in Table 7. Here, Optimal is absent

Table 7: Computing Times of the Algorithms (in Milliseconds)

| $\lvert\mathcal{S}\rvert$ \ $\lvert\mathcal{N}\rvert$ | GS-C | | | | | | GSP-C | | | | | | TeLa | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 25 | 50 | 75 | 100 | 125 | 150 | 25 | 50 | 75 | 100 | 125 | 150 | 25 | 50 | 75 | 100 | 125 | 150 |
| 50 | 5 | 8 | 16 | 16 | 16 | 31 | ≤3 | ≤3 | ≤3 | ≤3 | ≤3 | 5 | 5 | 9 | 12 | 15 | 19 | 22 |
| 100 | 15 | 13 | 31 | 43 | 46 | 63 | ≤3 | 5 | 9 | 10 | 16 | 18 | 30 | 49 | 73 | 95 | 117 | 140 |
| 150 | 47 | 34 | 50 | 63 | 66 | 94 | 7 | 10 | 15 | 24 | 34 | 47 | 95 | 158 | 223 | 301 | 340 | 406 |
| 200 | 79 | 47 | 63 | 85 | 94 | 110 | 16 | 30 | 47 | 58 | 78 | 93 | 206 | 393 | 520 | 679 | 843 | **958** |
| 250 | 187 | 119 | 93 | 94 | 125 | 141 | 31 | 78 | 125 | 157 | 203 | 250 | 370 | 735 | **945** | 1285 | 1545 | 1817 |
| 300 | 219 | 133 | 120 | 140 | 156 | 172 | 86 | 172 | 258 | 359 | 437 | 532 | 455 | 1366 | 1651 | 2190 | 2554 | 3105 |




(a) Impact of service number




(b) Impact of edge node number

Fig. 9: Simulation results under various service numbers and edge node numbers.

again due to its exponential computation complexity and extremely long computing times. Besides, as a randomized iterative algorithm, GS-C may search a small solution set and its computing time not surely scales with the whole solution space. For example, it spends an unexpectedly long time for 25 edges and 300 services, because the edges are now overloaded and it has to spend more time in finding a feasible solution in each iteration. Moreover, compared to GS-C and GSP-C, TeLa indeed spends a little more time in making finer-grained computation configuration decisions. Nevertheless, when the system has services less than 250 (or 200) and edges less than 75 (or 150), which covers many real-world scenarios, TeLa's computing time is within one second, showing that TeLa is time efficient.

## 6. Conclusion

With containers' fine-grained computation resource isolation, edges' heterogeneity and services' diversity taken into account, this paper jointly optimizes service placement and computation configuration for provisioning containerized services at edges. We propose a two-stage greedy and local-search combined algorithm TeLa based on convex and submodular optimization, and prove its approximation ratio and polynomial computation complexity. At last, we evaluate TeLa upon an edge computing prototype and the result verifies its superiority over state-of-the-art algorithms. In our future work, we will extend this joint problem to cover a wide area scenario, where edges in different serving regions could cooperate and workload scheduling should be incorporated.

## Appendix A. The Proof of Lemma 1

*Proof.* The first statement could be easily proved by combining the definition of $\boldsymbol{\mathcal{F}}$ and the constraints of $\mathscr{P}_2$. We omit its proof and focus on the second statement.

During this proof, we rewrite the objective function of $\mathscr{P}_1$ as $\Upsilon(\boldsymbol{x},\boldsymbol{y})+\Psi(\boldsymbol{x})$ for presentation brevity, where

$$\Upsilon(\boldsymbol{x},\boldsymbol{y})=\sum\nolimits_{s\in\mathcal{S}}\sum\nolimits_{n\in\mathcal{N}}x_{s,n}\times(\lambda_s d_n+\frac{\alpha_s\lambda_s}{y_s-\alpha_s\lambda_s}),$$
$$\Psi(\boldsymbol{x})=\sum\nolimits_{s\in\mathcal{S}}x_{s,o}\times(\lambda_s d_o+\frac{\alpha_s\lambda_s}{c_o}+\omega\beta_s\lambda_s).$$

Then, the objective function of $\mathscr{P}_3$ could be expressed as $\Phi(\boldsymbol{x})+\Psi(\boldsymbol{x})$, and according to $\mathscr{P}_2$, we have

$$\Phi(\boldsymbol{x})=\min\{\Upsilon(\boldsymbol{x},\boldsymbol{y})\mid\boldsymbol{y}\text{ is feasible to }\mathscr{P}_2\text{ under }\boldsymbol{x}\}. \tag{A.1}$$

To proceed, let $v_1^*$ be the optimal value of $\mathscr{P}_1$ and $\boldsymbol{x}_1^*$ with $\boldsymbol{y}_1^*$ be one of its optimal solutions. Let $v_3^*$ be the optimal value of $\mathscr{P}_3$ and $\boldsymbol{x}_3^*$ be one of its optimal solutions.

*1) proving $v_3^*\le v_1^*$.* We have the following inequations:

$$\begin{aligned}\Phi(\boldsymbol{x}_3^*)+\Psi(\boldsymbol{x}_3^*)&\le\Phi(\boldsymbol{x}_1^*)+\Psi(\boldsymbol{x}_1^*)\\&\le\Upsilon(\boldsymbol{x}_1^*,\boldsymbol{y}_1^*)+\Psi(\boldsymbol{x}_1^*).\end{aligned}$$

The first inequality holds because $\mathscr{P}_3$ is a minimization problem and $\boldsymbol{x}_3^*$ is its optimal solution. Meanwhile, Eq. (A.1) guarantees that for any feasible solution $\boldsymbol{x}$ and $\boldsymbol{y}$,

we have $\Phi(\boldsymbol{x}) \leq \Upsilon(\boldsymbol{x}, \boldsymbol{y})$. Therefore, the second inequality holds. Combining these inequations with $v_3^* = \Phi(\boldsymbol{x_3^*}) + \Psi(\boldsymbol{x_3^*})$ and $v_1^* = \Upsilon(\boldsymbol{x_1^*}, \boldsymbol{y_1^*}) + \Psi(\boldsymbol{x_1^*})$ yields $v_3^* \leq v_1^*$.

*2) proving $v_1^* \leq v_3^*$.* According to the first statement of this lemma, $\boldsymbol{x}_3^*$ with $\boldsymbol{y}^{\boldsymbol{x}_3^*}$ is also a feasible solution to $\mathscr{P}_1$. As a consequence, we have

$$\begin{aligned}\Upsilon(\boldsymbol{x_1^*}, \boldsymbol{y_1^*}) + \Psi(\boldsymbol{x_1^*}) &\leq \Upsilon(\boldsymbol{x_3^*}, \boldsymbol{y}^{\boldsymbol{x}_3^*}) + \Psi(\boldsymbol{x_3^*}) \\ &= \Phi(\boldsymbol{x_3^*}) + \Psi(\boldsymbol{x_3^*}).\end{aligned}$$

The inequality holds due to the optimality of $\boldsymbol{x_1^*}$ and $\boldsymbol{y_1^*}$ for $\mathscr{P}_1$. The equality holds since $\boldsymbol{y}^{\boldsymbol{x}_3^*}$ is the optimal solution to $\mathscr{P}_2$. Then similar to the first case, we get $v_1^* \leq v_3^*$.

Putting $v_3^* \leq v_1^*$ and $v_1^* \leq v_3^*$ together, we have $v_1^* = v_3^*$, which means $\mathscr{P}_3$ has the same optimal value with $\mathscr{P}_1$. □

## Appendix B. The Proof of Lemma 2

*Proof.* To prove this lemma, we consider any two subsets $\mathcal{X} \subseteq \mathcal{X}' \subseteq \mathcal{S} \times \mathcal{N}$ and any element $(e, n) \in \mathcal{S} \times \mathcal{N} \setminus \mathcal{X}'$, such that $\mathcal{X}$, $\mathcal{X}'$, $\mathcal{X} \cup \{(e,n)\}$ and $\mathcal{X}' \cup \{(e,n)\}$ are all feasible. Then, the margin gains at $\mathcal{X}$ and $\mathcal{X}'$ are

$$\begin{aligned}&\Omega(\mathcal{X} \cup \{(e,n)\}) - \Omega(\mathcal{X}) \\ =&\lambda_e(d_o - d_n) + \frac{\mu_e}{c_o} + \omega\beta_e\lambda_e - \frac{\mu_e + 2\sqrt{\mu_e}(\sum_{s\in\mathcal{S}_{\mathcal{X},n}}\sqrt{\mu_s})}{C_n - (\sum_{s\in\mathcal{S}_{\mathcal{X},n}}\mu_s) - \mu_e} \\ &- \frac{\mu_e(\sum_{s\in\mathcal{S}_{\mathcal{X},n}}\sqrt{\mu_s})^2}{(C_n - (\sum_{s\in\mathcal{S}_{\mathcal{X},n}}\mu_s))(C_n - (\sum_{s\in\mathcal{S}_{\mathcal{X},n}}\mu_s) - \mu_e)}, \\ &\Omega(\mathcal{X}' \cup \{(e,n)\}) - \Omega(\mathcal{X}') \\ =&\lambda_e(d_o - d_n) + \frac{\mu_e}{c_o} + \omega\beta_e\lambda_e - \frac{\mu_e + 2\sqrt{\mu_e}(\sum_{s\in\mathcal{S}_{\mathcal{X}',n}}\sqrt{\mu_s})}{C_n - (\sum_{s\in\mathcal{S}_{\mathcal{X}',n}}\mu_s) - \mu_e} \\ &- \frac{\mu_e(\sum_{s\in\mathcal{S}_{\mathcal{X}',n}}\sqrt{\mu_s})^2}{(C_n - (\sum_{s\in\mathcal{S}_{\mathcal{X}',n}}\mu_s))(C_n - (\sum_{s\in\mathcal{S}_{\mathcal{X}',n}}\mu_s) - \mu_e)}.\end{aligned}$$

Two insights are found from the above equations. First, as $\mathcal{X}$ is a subset of $\mathcal{X}'$, we have for any edge node $n \in \mathcal{N}$, $\mathcal{S}_{\mathcal{X},n} \subseteq \mathcal{S}_{\mathcal{X}',n}$. Then the following inequalities hold:

$$\begin{aligned}\sum\nolimits_{s\in\mathcal{S}_{\mathcal{X},n}}\mu_s &\leq \sum\nolimits_{s\in\mathcal{S}_{\mathcal{X}',n}}\mu_s, \\ \sum\nolimits_{s\in\mathcal{S}_{\mathcal{X},n}}\sqrt{\mu_s} &\leq \sum\nolimits_{s\in\mathcal{S}_{\mathcal{X}',n}}\sqrt{\mu_s}.\end{aligned}$$

Thus, we have $\Omega(\mathcal{X}\cup\{(e,n)\}) - \Omega(\mathcal{X}) \geq \Omega(\mathcal{X}'\cup\{(e,n)\}) - \Omega(\mathcal{X}')$, which proves the submodularity of $\Omega(\mathcal{X})$. Second, it is easy to see that $\Omega(\mathcal{X} \cup \{(e,n)\}) - \Omega(\mathcal{X})$ is not surely positive or negative. Intuitively, a large set of $\mathcal{X}$ tends to have negative margin gains since the edge nodes are now overloaded, and vice versa. As a consequence, $\Omega(\mathcal{X})$ is non-monotone. □

## Appendix C. The Proof of Lemma 3

*Proof.* Obviously, an empty set is a subset of $\boldsymbol{\mathcal{F}}$, since we could place all services on the cloud. Moreover, according to the constraints of $\mathscr{P}_5$, a subset $\mathcal{X}$ of a feasible solution $\mathcal{X}' \in \boldsymbol{\mathcal{F}}$ is also feasible, implying $\mathcal{X} \in \boldsymbol{\mathcal{F}}$. Thus, $(\mathcal{S} \times \mathcal{N}, \boldsymbol{\mathcal{F}})$ is an independence system. As for determining $p$, we first have $\max_{\mathcal{T}\subseteq\mathcal{S}\times\mathcal{N},|\mathcal{T}|\leq 1}\frac{r(\mathcal{T})}{\rho(\mathcal{T})} = 1$, where $r(\mathcal{T})$ and $\rho(\mathcal{T})$ is the cardinality of the largest and the smallest basis of $\mathcal{T}$, respectively, since currently there is at most one service involved in $\mathcal{T}$. Then we focus on any subset whose cardinality is larger than one, i.e., $\mathcal{T} \subseteq \mathcal{S}\times\mathcal{N}, |\mathcal{T}| > 1$. Since we could place at least one service at edges in practical scenarios, we have $\min\rho(\mathcal{T}) \geq 1$. Moreover, as each service could be placed on at most one node, the cardinality of any independent set of $\mathcal{T}$ is bounded by the service number, leading to $\max r(\mathcal{T}) \leq |\mathcal{S}|$. Meanwhile, the limited resources of edges also restrict the number of edge-hosted services. Precisely, the number of services that each edge node could host is at most $\min\{\frac{\max_{n\in\mathcal{N}}C_n}{\min_{s\in\mathcal{S}}\mu_s}, \frac{\max_{n\in\mathcal{N}}M_n}{\min_{s\in\mathcal{S}}m_s}, \frac{\max_{n\in\mathcal{N}}G_n}{\min_{s\in\mathcal{S}}g_s}, \frac{\max_{n\in\mathcal{N}}B_n}{\min_{s\in\mathcal{S}}\lambda_s\beta_s}\}$. Summarizing these cases, we finally have $\max_{\mathcal{T}\subseteq\mathcal{S}\times\mathcal{N}}\frac{r(\mathcal{T})}{\rho(\mathcal{T})} \leq \min\{|\mathcal{N}|(\frac{\max_{n\in\mathcal{N}}C_n}{\min_{s\in\mathcal{S}}\mu_s}, \frac{\max_{n\in\mathcal{N}}M_n}{\min_{s\in\mathcal{S}}m_s}, \frac{\max_{n\in\mathcal{N}}G_n}{\min_{s\in\mathcal{S}}g_s}, \frac{\max_{n\in\mathcal{N}}B_n}{\min_{s\in\mathcal{S}}\lambda_s\beta_s}), |\mathcal{S}|\}$. Following Definition 2 yields the expression of $p$. □

## References


[1] M. K. Hussein, M. H. Mousa, M. A. Alqarni, A placement architecture for a container as a service (CaaS) in a cloud environment, Journal of Cloud Computing 8 (1) (2019) 7. doi:10.1186/s13677-019-0131-1.
URL https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-019-0131-1

[2] V. Liagkou, G. Fragiadakis, E. Filiopoulou, C. Michalakelis, T. Kamalakis, M. Nikolaidou, A pricing model for Container-as-a-Service, based on hedonic indices, Simulation Modelling Practice and Theory 115 (2022) 102441. doi:10.1016/j.simpat.2021.102441.
URL https://www.sciencedirect.com/science/article/pii/S1569190X21001362

[3] A. Saboor, M. F. Hassan, R. Akbar, S. N. M. Shah, F. Hassan, S. A. Magsi, M. A. Siddiqui, Containerized Microservices Orchestration and Provisioning in Cloud Computing: A Conceptual Framework and Future Perspectives, Applied Sciences 12 (12) (2022) 5793, number: 12 Publisher: Multidisciplinary Digital Publishing Institute. doi:10.3390/app12125793.
URL https://www.mdpi.com/2076-3417/12/12/5793

[4] Amazon Elastic Container Service (Amazon ECS).
URL https://aws.amazon.com/ecs/

[5] Google Kubernetes Engine (GKE).
URL https://cloud.google.com/kubernetes-engine

[6] Azure Container Instances.
URL https://azure.microsoft.com/en-us/services/container-instances/

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge Computing: Vision and Challenges, IEEE Internet of Things Journal 3 (5) (2016) 637–646, conference Name: IEEE Internet of Things Journal. doi:10.1109/JIOT.2016.2579198.

[8] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39, publisher: IEEE.

[9] C. Pahl, S. Helmer, L. Miori, J. Sanin, B. Lee, A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters, in: 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), 2016, pp. 117–124. doi:10.1109/W-FiCloud.2016.36.

[10] P. Bellavista, A. Zanni, Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi, in: Proceedings of the 18th International Conference on Distributed Computing and Networking, ICDCN '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1–10.

doi:10.1145/3007748.3007777.
URL https://doi.org/10.1145/3007748.3007777

[11] L. Gu, D. Zeng, J. Hu, B. Li, H. Jin, Layer Aware Microservice Placement and Request Scheduling at the Edge, in: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, 2021, pp. 1–9, iSSN: 2641-9874. doi:10.1109/INFOCOM42981.2021.9488779.

[12] L. Pan, L. Wang, S. Chen, F. Liu, Retention-Aware Container Caching for Serverless Edge Computing, in: IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, 2022, pp. 1069–1078, iSSN: 2641-9874. doi:10.1109/INFOCOM48880.2022.9796705.

[13] P. Kayal, Kubernetes in Fog Computing: Feasibility Demonstration, Limitations and Improvement Scope : Invited Paper, in: 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), 2020, pp. 1–6. doi:10.1109/WF-IoT48130.2020.9221340.

[14] Kubernetes: Production-Grade Container Orchestration (2022).
URL https://kubernetes.io/

[15] MicroK8s - Zero-ops Kubernetes for developers, edge and IoT.
URL https://microk8s.io/

[16] KubeEdge, KubeEdge.
URL https://kubeedge.io/

[17] K3s: Lightweight Kubernetes.
URL https://k3s.io/

[18] Alibaba cluster data.
URL https://github.com/alibaba/clusterdata

[19] Huawei Cloud.
URL https://www.huaweicloud.com/pricing/calculator.html#/ecs

[20] Google Cloud.
URL https://cloud.google.com/vpc/network-pricing

[21] AWS Cloud.
URL https://aws.amazon.com/ec2/pricing/on-demand/

[22] D. Palomar, M. Chiang, A tutorial on decomposition methods for network utility maximization, IEEE Journal on Selected Areas in Communications 24 (8) (2006) 1439–1451, conference Name: IEEE Journal on Selected Areas in Communications. doi:10.1109/JSAC.2006.879350.

[23] S. Pasteris, S. Wang, M. Herbster, T. He, Service placement with provable guarantees in heterogeneous edge computing systems, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 514–522.

[24] T. Ouyang, Z. Zhou, X. Chen, Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing, IEEE JSAC 36 (10) (2018) 2333–2345.

[25] G. Lia, M. Amadeo, G. Ruggeri, C. Campolo, A. Molinaro, V. Loscrì, In-network placement of delay-constrained computing tasks in a softwarized intelligent edge, Computer Networks 219 (2022) 109432. doi:10.1016/j.comnet.2022.109432.
URL https://www.sciencedirect.com/science/article/pii/S1389128622004662

[26] J. Dou, F. Yuan, J. Cao, X. Meng, X. Ma, Z. Guo, Placement Combination between Heterogeneous Services and Heterogeneous Capacitated Servers in Edge Computing, Journal of Grid Computing 21 (1) (2023) 16. doi:10.1007/s10723-023-09644-3.
URL https://doi.org/10.1007/s10723-023-09644-3

[27] J. Xu, L. Chen, P. Zhou, Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 207–215. doi:10.1109/INFOCOM.2018.8485977.

[28] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, L. Tassiulas, Joint service placement and request routing in multi-cell mobile edge computing networks, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 10–18.

[29] T. He, H. Khamfroush, S. Wang, T. La Porta, S. Stein, It's Hard to Share: Joint Service Placement and Request Scheduling in Edge Clouds with Sharable and Non-Sharable Resources, in: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 2018, pp. 365–375, iSSN: 2575-8411. doi:10.1109/ICDCS.2018.00044.

[30] V. Farhadi, F. Mehmeti, T. He, T. La Porta, H. Khamfroush, S. Wang, K. S. Chan, Service Placement and Request Scheduling for Data-intensive Applications in Edge Clouds, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 1279–1287.

[31] F. Wang, C. Zhang, J. Liu, Y. Zhu, H. Pang, L. Sun, Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized QoE, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 910–918.

[32] D. Lee, Y. Kim, M. Song, Cost-Effective, Quality-Oriented Transcoding of Live-Streamed Video on Edge-Servers, IEEE Transactions on Services Computing 16 (4) (2023) 2503–2516, conference Name: IEEE Transactions on Services Computing. doi:10.1109/TSC.2023.3256425.

[33] X. Ma, A. Zhou, S. Zhang, S. Wang, Cooperative Service Caching and Workload Scheduling in Mobile Edge Computing, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020, pp. 2076–2085, iSSN: 2641-9874. doi:10.1109/INFOCOM41043.2020.9155455.

[34] T. Cao, Y. Jin, X. Hu, S. Zhang, Z. Qian, B. Ye, S. Lu, Adaptive provisioning for mobile cloud gaming at edges, Computer Networks 205 (2022) 108704. doi:10.1016/j.comnet.2021.108704.
URL https://www.sciencedirect.com/science/article/pii/S138912862100565X

[35] T. Goethals, F. De Turck, B. Volckaert, FLEDGE: Kubernetes Compatible Container Orchestration on Low-Resource Edge Devices, in: C.-H. Hsu, S. Kallel, K.-C. Lan, Z. Zheng (Eds.), Internet of Vehicles. Technologies and Services Toward Smart Cities, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2020, pp. 174–189. doi:10.1007/978-3-030-38651-1_16.

[36] T. Ouyang, R. Li, X. Chen, Z. Zhou, X. Tang, Adaptive User-managed Service Placement for Mobile Edge Computing: An Online Learning Approach, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 1468–1476.

[37] C. Li, Q. Zhang, C. Huang, Y. Luo, Optimal Service Selection and Placement Based on Popularity and Server Load in Multi-access Edge Computing, Journal of Network and Systems Management 31 (1) (2022) 15. doi:10.1007/s10922-022-09703-2.
URL https://doi.org/10.1007/s10922-022-09703-2

[38] Z. Xiang, S. Deng, F. Jiang, H. Gao, J. Tehari, J. Yin, Computing Power Allocation and Traffic Scheduling for Edge Service Provisioning, in: 2020 IEEE International Conference on Web Services (ICWS), 2020, pp. 394–403. doi:10.1109/ICWS49710.2020.00058.

[39] Q. Fan, N. Ansari, Application Aware Workload Allocation for Edge Computing-Based IoT, IEEE Internet of Things Journal 5 (3) (2018) 2146–2153, conference Name: IEEE Internet of Things Journal. doi:10.1109/JIOT.2018.2826006.
URL https://ieeexplore.ieee.org/document/8336866

[40] A. Al-Shuwaili, O. Simeone, Energy-Efficient Resource Allocation for Mobile Edge Computing-Based Augmented Reality Applications, IEEE Wireless Communications Letters 6 (3) (2017) 398–401, conference Name: IEEE Wireless Communications Letters. doi:10.1109/LWC.2017.2696539.
URL https://ieeexplore.ieee.org/document/7906521/authors#authors

[41] Y. Nam, Y. Choi, B. Yoo, H. Eom, Y. Son, EdgeIso: Effective Performance Isolation for Edge Devices, in: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020, pp. 295–305, iSSN: 1530-2075. doi:10.1109/IPDPS47924.2020.00039.
URL https://ieeexplore.ieee.org/abstract/document/9139806

[42] V. Maniezzo, M. A. Boschetti, T. Stützle, The Generalized Assignment Problem, in: V. Maniezzo, M. A. Boschetti, T. Stützle (Eds.), Matheuristics: Algorithms and Implementations, EURO

Advanced Tutorials on Operational Research, Springer International Publishing, Cham, 2021, pp. 3–33. doi:10.1007/978-3-030-70277-9_1.
URL https://doi.org/10.1007/978-3-030-70277-9_1

[43] Docker Engine.
URL https://www.docker.com/

[44] F. A. Salaht, F. Desprez, A. Lebre, An Overview of Service Placement Problem in Fog and Edge Computing, ACM Computing Surveys 53 (3) (2020) 65:1–65:35. doi:10.1145/3391196.
URL https://dl.acm.org/doi/10.1145/3391196

[45] B. Sonkoly, J. Czentye, M. Szalay, B. Németh, L. Toka, Survey on Placement Methods in the Edge and Beyond, IEEE Communications Surveys & Tutorials 23 (4) (2021) 2590–2629, conference Name: IEEE Communications Surveys & Tutorials. doi:10.1109/COMST.2021.3101460.
URL https://ieeexplore.ieee.org/document/9502167

[46] H. Tabatabaee Malazi, S. R. Chaudhry, A. Kazmi, A. Palade, C. Cabrera, G. White, S. Clarke, Dynamic Service Placement in Multi-Access Edge Computing: A Systematic Literature Review, IEEE Access 10 (2022) 32639–32688, conference Name: IEEE Access. doi:10.1109/ACCESS.2022.3160738.
URL https://ieeexplore.ieee.org/document/9738624

[47] Z. Ma, S. Zhang, Z. Chen, T. Han, Z. Qian, M. Xiao, N. Chen, J. Wu, S. Lu, Towards Revenue-Driven Multi-User Online Task Offloading in Edge Computing, IEEE Transactions on Parallel and Distributed Systems 33 (5) (2022) 1185–1198, conference Name: IEEE Transactions on Parallel and Distributed Systems. doi:10.1109/TPDS.2021.3105325.
URL https://ieeexplore.ieee.org/document/9516964

[48] L. Chen, S. Zhou, J. Xu, Computation Peer Offloading for Energy-Constrained Mobile Edge Computing in Small-Cell Networks, IEEE/ACM Transactions on Networking 26 (4) (2018) 1619–1632, conference Name: IEEE/ACM Transactions on Networking. doi:10.1109/TNET.2018.2841758.

[49] V. Sundarapandian, Probability, statistics and queuing theory, PHI Learning Pvt. Ltd., 2009.

[50] S. Sundar, B. Liang, Offloading Dependent Tasks with Communication Delay and Deadline Constraint, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 37–45. doi:10.1109/INFOCOM.2018.8486305.

[51] S. Boyd, S. P. Boyd, L. Vandenberghe, Convex optimization, Cambridge university press, 2004.

[52] G. L. Nemhauser, L. A. Wolsey, M. L. Fisher, An analysis of approximations for maximizing submodular set functions—I, Mathematical Programming 14 (1) (1978) 265–294. doi:10.1007/BF01588971.
URL https://doi.org/10.1007/BF01588971

[53] A. Gupta, A. Roth, G. Schoenebeck, K. Talwar, Constrained Non-monotone Submodular Maximization: Offline and Secretary Algorithms, in: A. Saberi (Ed.), Internet and Network Economics, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2010, pp. 246–257. doi:10.1007/978-3-642-17572-5_20.

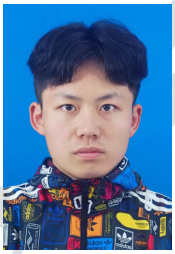
[54] U. Feige, V. S. Mirrokni, J. Vondrák, Maximizing Non-monotone Submodular Functions, SIAM Journal on Computing 40 (4) (2011) 1133–1153, publisher: Society for Industrial and Applied Mathematics. doi:10.1137/090779346.
URL https://epubs.siam.org/doi/abs/10.1137/090779346

[55] A. Samanta, Z. Chang, Adaptive Service Offloading for Revenue Maximization in Mobile Edge Computing With Delay-Constraint, IEEE Internet of Things Journal 6 (2) (2019) 3864–3872, conference Name: IEEE Internet of Things Journal. doi:10.1109/JIOT.2019.2892398.

[56] H. Tan, Z. Han, X.-Y. Li, F. C. Lau, Online job dispatching and scheduling in edge-clouds, in: IEEE INFOCOM 2017-IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9.

**Tuo Cao** received the B.S. degree from the Department of Computer Science and Technology, Xian Jiaotong University, in 2019. He is currently working towards the Ph.D. degree under the supervision of Professor Zhuzhong Qian in Nanjing University, China. His research interests include edge computing, distributed systems, scheduling algorithms and optimization theory. To date, his research has been published in journals such as Computer Networks, and in conferences such as IWQoS, WoWMoM, MSN. He received the Cisco best paper candidate award from WoWMoM 2021.

**Qinhui Wang** received his B.S., M.S., and Ph.D. degrees from Nanjing University in 2007, 2011, and 2015, respectively, all in computer science. He is now with Department of Military Training and Management, Army Command College, China. His current research interests include Mobile Wireless Networks, Cloud Computing and Edge Computing.



**Yuhan Zhang** received the BS degree from the School of Computer Science and Engineering, Nanjing University of Science and Technology, in 2021. He is currently pursuing the MS degree under the supervision of Professor Zhuzhong Qian in the Department of Computer Science and Technology, Nanjing University. His research interests include edge computing, service placement, and intelligent inference.

**Zhuzhong Qian** is currently a full professor at the Department of Computer Science and Technology, and member of National Key Laboratory for Novel Software Technology, Nanjing University, P. R. China. He received his PhD. Degree in 2007. His research interests include cloud computing, edge computing, and distributed machine learning. He is the chief member of several national research projects on cloud computing and edge computing. His research has been published in journals such as TPDS, TON, TC, and TMC, and in conferences such as INFOCOM, ICDCS, SECON, and IPDPS. He received best paper awards from IMIS 2013, ICA3PP 2014 and APNet 2018.

**Yue Zeng** received the M.S. degree in the department of electronic information engineering from Southwest University, Chongqing, China, in 2019. He is currently working toward the Ph.D. degree in the department of computer science and technology in Nanjing University, China. His research interests include federated learning, deep reinforcement learning, network functions virtualization, distributed computing, and edge comput-

ing. He has published over ten papers in relevant journals and conferences, including IEEE Transactions on Service Computing (TSC), IEEE Transactions on Communications(TCOM), IEEE Transactions on Cloud Computing(TCC), and Computer Networks (COMNET), etc.

**Mingtao Ji** received the B.E. degree from the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics of in 2018. He is currently pursuing the PhD degree under the supervision of Professor Zhuzhong Qian in Nanjing University. To date, he has already published over 7 papers, including in journals such as IEEE TON, Electric Power ICT, and in conferences such as IEEE ICC, IEEE INFOCOM, IEEE ISPA. His research interests include P4 switch, big data analytics and distributed machine learning.

**Hesheng Sun** received the BS degree from the Department of Computer Science and Technology, Xi'an Jiaotong University in 2020. He is currently pursuing the PhD degree under the supervision of Professor Zhuzhong Qian in Nanjing University. He was a visiting student with the University of Alberta, Canada in 2018. His research interests include machine learning and edge computing.

**Baoliu Ye** is a full professor at Department of Computer Science and Technology, Nanjing University. He received his Ph.D. in computer science from Nanjing University, China in 2004. He served as a visiting researcher of the University of Aizu, Japan from March 2005 to July 2006, and the Dean of School of Computer and Information, Hohai University since January 2018. His current research interests mainly include distributed systems, cloud computing, wireless networks with over 70 papers published in major conferences and journals. Prof. Ye served as the TPC co-chair of HotPOST12, Hot-POST11, P2PNet10. He is the regent of CCF, the Secretary-General of CCF Technical Committee of Distributed Computing and Systems.

**CRediT Authorship Contribution Statement**

**Tuo Cao**: Conceptualization, Methodology, Formal Analysis, Software, Writing - Original Draft

**Qinhui Wang**: Methodology, Validation, Writing - Review & Editing

**Yuhan Zhang**: Methodology, Validation, Writing - Review & Editing

**Zhuzhong Qian**: Supervision, Project Administration

**Yue Zeng**: Methodology, Validation, Writing - Review & Editing

**Mingtao Ji**: Writing - Review & Editing

**Hesheng Sun**: Writing - Review & Editing

**Baoliu Ye**: Supervision, Validation

**Declaration of interests**

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

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: