ASAP: Adaptive and Scalable Microservice Provisioning for Edge-IoT Networks

Amit Samanta[1] Flavio Esposito[2] Tri Gia Nguyen[3]
[1]*University of Utah, USA* [2]*Saint Louis University, USA* [3]*FPT University, Vietnam*

***Abstract*—The edge computing paradigm aims at provisioning compute and storage resources from Internet-of-Things (IoT)-enabled devices at the network edge, while disseminating the end-to-end latency and increasing mobile computational capacity. For scalable provisioning of microservices, recently proposed methods frequently need a priori information on the microservice type, computational capacity, and other parameters. In the presence of such restrictions, the present methods need to face lower quality of experience for mission-critical and resource-hungry applications. In this paper, we study the problem of microservice provisioning for mobile edge computing, and we propose an adaptive and scalable solution. The core of our optimal microservice differentiation scheme is ASAP, a microservice-level abstraction for the orchestration of network resources. ASAP provides adaptive and scalable microservice provisioning by minimizing microservice delay, while maximizing the network throughput. With a prototype tested over a local testbed and trace-driven simulations, we show how ASAP increases throughput compared to other solutions based on the executions of microservices at the edge.**

I. Introduction

In recent years, Mobile Edge Computing (MEC) has become an important paradigm in the era of Internet-of-Things (IoT) [1]. IoT technologies, among others, have introduced the need for latency-sensitive management strategies to ensure reliable performance in resource-constrained environments. An example of such a situation is a man-made or natural disaster incident, where first responders operate in areas with limited computational and network resources. One of the core edge computing mechanisms is *provisioning*, $i.e.$, outsourcing computational loads of multiple network functions to a server located at the network's edge. Aside from latency-sensitive applications, the mobile edge computing paradigm has shown profit improvements for mobile network operators and edge devices by increasing resource utilization while incorporating microservices. By integrating mobile devices with nearby computational capabilities, microservice providers have been able to support a plethora of new applications [2], [3]. A shared goal of such MEC applications is efficiency: while provisioning real-time microservices to minimize overall delay and maximize the network throughput at the network edge.

Motivation. However, the design of an adaptive microservice provisioning scheme in MEC [4] is challenging. Mobile applications are generally resource-sensitive and demand-agnostic, which require intensive computational power. On the other way, mobile devices, however, typically have tight computational resources and limited battery life. Moreover, such challenges are exacerbated by the wireless radio network inefficiencies that inevitably hinder the performance of the microservice provisioning phase. One inefficiency comes from the lack of proper interference management. As the density of mobile devices increases at the proximity of same wireless medium when attempting to provision, mutual- and cross-technology interference become harder to manage; this causes the microservice provisioning rate to decrease, and in turn, an increase in energy consumption and microservice delay. This paper proposes a microservice provisioning scheme aiming to mitigate the following challenges:

- Due to the unavailability of an adaptive and flexible microservice provisioning scheme, how do we cope with the increase of microservice delay for different applications, and with inefficiencies introduced by load modifications and edge function dependencies?
- Given suboptimal microservice differentiation, cope with the mix of heterogeneous microservices at the edge. By heterogeneous, we mean *delay-critical*, *delay-tolerant or normal*, and *running in background*.
- Cope with microservice delay and bandwidth requirements dynamicity due to a non-stationary environment.

Our Contributions. Various schemes to provision microservices at the network edge exist to solve the above-described challenges. Existing provisioning solutions assume that all provisioning requests coming from mobile devices have an equal level of latency sensitivity; in this work, we remove such assumptions introducing the possibility of providing a mixture of hybrid services, $i.e.$, a cloud capable of hosting both cloud- and micro-services. We show how this microservice differentiation scheme improves the efficiency of the provisioning mechanism. We design, prototype, and evaluate an adaptive and scalable microservice provisioner that allocates resources fairly to microservices according to their demands and latency sensitiveness. In particular:

- We propose an optimal microservice differentiation scheme for different traffic flows to get an optimal success rate for edge devices. Our scheme is adaptive to different types of traffic flows and their priority levels and maximizes the throughput in the network while minimizing the microservice delay.

 ISBN: 978-3-903176-56-0

- To establish the practicality of our approach, we implement our architecture ASAP (**A**daptive and **Sc**A**l**able Microservice **P**rovisioning), and test its scalability and microservice utilization performance over a local testbed and also through simulation results.
- Our results show that ASAP significantly improves fairness and network throughput over the existing schemes.

II. Related Work

Solutions tackling microservice provisioning at the network edge have increased, given the need to minimize the provisioning delay and maximize the utilization.

Service Provisioning at Edge In [5], Deng *et al.* investigates the cache mechanism with composite services in MEC. The authors present a service cache policy by proposing a consumption-driven searching algorithm to improve the service provision systems' performance. In [6], Ma *et al.* investigates the resource provisioning problem to provide guaranteed QoS with minimal cost. Zhou *et al.* [7] studied the workload offloading problem for user requirements in vehicular networks with limited battery capacity. Cao *et al.* [8] designed a service provisioning model considering the federal architecture of edge platforms. Zhang *et al.* [9] considered a latency-aware service provisioning scheme for IoT-enabled edge platforms. Gu *et al.* [10] proposed a service provisioning and scheduling scheme for a blockchain-enabled edge platforms. Li *et al.* [11] designed a cooperative service provisioning scheme using a data-driven approach while considering the demand uncertainty of edge services.

Advances in Microservice Applications Unlike the edge service provisioning scheme, here we also discuss existing works on microservice-enabled platforms. Filip *et al.* [12] proposed a microservice scheduling algorithm for IoT-enabled heterogeneous edge computing applications. Zhao *et al.* [13] designed a distributed redundancy scheduling mechanism for microservice-based applications at the edge. On the other hand, Chen *et al.* [14] proposed a microservice deployment scheme for an IoT-enabled hybrid platform following the reinforcement learning approach. Wang *et al.* [15] designed a microservice placement approach for a collaborative edge platform. On the other hand, Samanta *et al.* [16] proposed a dynamic microservice scheduling mechanism for an IoT-enabled edge computing platform. Similarly, Samanta *et al.* [17] proposed a latency-optimal heuristic scheduling for microservice-enabled edge computing platform.

Drawbacks of Prior Solutions. Some existing solutions provide schemes for the microservices at the network edge. However, they mostly assumed that the microservices coming from mobile devices are only microservices, but in real life, they could be a mix of both cloud and microservices. They sidelined this aspect. Therefore, it is necessary to differentiate the edge and cloud microservices to provision them first while minimizing the microservice delay efficiently. We also need to design an adaptive and scalable microservice provisioner to provide fair resources to microservices and increase the network throughput.

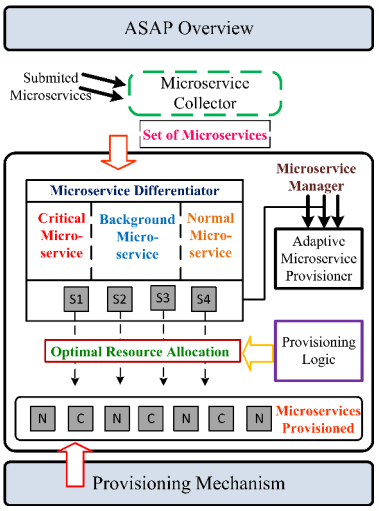


Figure 1: Architectural view of ASAP

III. ASAP Overview

To clarify our proposed approach, consider the scenario, where a pool of microservices is collected from different mobile edge device applications. Such applications are typically composed of different heterogeneous microservices. The microservice pool is composed of different microservices – *delay-critical*, *delay non-critical or normal* and *delay-tolerant or background**. Note that the edge infrastructure does not have any prior knowledge about the microservice or application type. Note that delay-critical microservices are to be provisioned and require their computational resources to be executed at mobile edge servers as soon as possible. The only parameter known to the edge infrastructure is the delay requirement of each microservice. To this end, we design an optimal *differentiator process*, whose job differentiates the microservices into three microservice classes – microservice class 1 for latency-critical microservices, microservice class 2 for non-latency critical or normal microservices, microservice class 3 for background or delay-tolerant microservices. Figure 2 shows the basic responsibility of the microservice differentiator. This section overviews our proposed schema. When designing ASAP†, we aimed to decouple the rigidity between different types of microservices according to their application requirements. Most Mobile Edge Computing (MEC) systems aim to deliver latency-sensitive microservices while maximizing their throughput. Many edge computing systems optimally differentiate their offered

*Microservices may have different levels of sensitiveness to delay, for example, delay-critical applications may be augmented, virtual reality or haptic devices, non-critical delay application may be (meta-data or sensorial data generator, while delay-tolerant microservices may be background processes such as system updates or logging.

†ASAP is designed to place as middleware at the network edge. Here, the microservices originate from applications on smartphones/devices.

microservices based on application types ($i.e.$, critical or normal). The microservices are provisioned at the edge or cloud based on their application types. The key idea behind ASAP is to use the maximal bandwidth required for these critical microservices to complete just before their deadlines, thus leaving minimal bandwidth to type-normal microservices to optimize their delay.

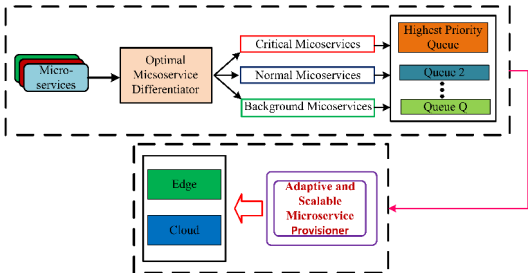


Figure 2: Overview of microservice differentiation

We now discuss the overall architectural view of ASAP (Figure 1). The first step is a discovery: microservices are collected from different applications running on different mobile devices. The collected set of microservices is sent to our designed microservice differentiator for an optimal microservice classification. We modeled three types of microservice classes — microservice class 1, designed for critical microservices, microservice class 2, designed for normal microservices, and microservice class 3, designed for background microservices. These are represented as s1, s2, and s3 in Figure 1. After completing our microservice differentiation step, the microservice classes are sent to a multilevel prioritization scheme, and priorities are set for each available microservice class. Finally, the prioritized microservices are sent to the adaptive microservice provisioner. The microservices are then provisioned to the edge of the cloud platform using the provisioning logic.

IV. Microservice Differentiation

In this section, we present the design of our optimal microservice differentiation scheme. As shown in Figure 2, this is used by our ASAP architecture to differentiate the microservices from a microservice pool with several heterogeneous microservices. By heterogeneous, we mean that each microservices has different resources and computational requirements. The types of microservices are unknown to ASAP, $i.e.$, ASAP does not have any prior knowledge about each microservice type.

A. System Design

Let us consider an edge computing environment and a list of real-time applications with heterogeneous characteristics. We assume that each application has a different and unique priority. Our main goal is to identify the real-time edge applications from a set of heterogeneous applications and assign them the highest provisioning priority. Let us suppose that the edge computing infrastructure may host a set of N microservices [18]. Each of these microservices may adopt a different priority Θ_i for all microservices. We associate with each microservice $i \in N$, a microservice delay requirement d_i. Each microservice's delay requirements may differ for each application; microservices with stringent delay requirements need to be processed at the network edge. We denote $\Theta \triangleq \{\ \Theta_i : i \in N\}$ and $\mathsf{D} \triangleq \{\ d_i : i \in N\}$ as priority and delay-requirement vectors, respectively.

As the property of each microservice is assigned, ASAP will choose from the microservice pool $\{\ (\Theta_i, d_i) : i \in N\}$ according to each microservice property and application. We model the microservices of mobile devices by their application types $\mathcal{A}$, where the application types are dependent on the microservice class $\mathcal{A} \in \mathcal{C}$. We assume the microservice class $\mathcal{C}$ to be made either edge or cloud: $\mathcal{C} = [+1, -1]$. Moreover, we denote with $f(\mathcal{A})$ and $F(\mathcal{A})$ the probability density and cumulative distribution functions of the application type. We assume that $f(\mathcal{A})$ is continuously differentiable and strictly positive over $\mathcal{C}$. When an application type chooses its microservice class $(\Theta; d)$, the utility is:

$$\mathcal{W}(\Theta, d; \mathcal{A}) = \mathcal{V} - \mathcal{A}\mathcal{S}(\Theta) - d \tag{1}$$

Each of the mathematical interpretations is defined as:

- The reward function $\mathcal{V}$, independent from the application type, represents the benefit of each edge device for accessing the microservices.
- $\mathcal{S}(\cdot)$ represents a utility discount function, which depends on the congestion level of the microservices. This discount function is designed to capture the congestion factor's negative effect on edge devices' performance.
- $\mathcal{S}(\Theta)$ is the congestion factor faced by different applications and their microservice type. To minimize its effect, the achieved value $\mathcal{S}(\Theta)$ is deducted from the reward function. Without loss of generality, we normalize the range of congestion factor to the interval $\mathcal{S}(\cdot) \in [0; 1]$. This scheme stated that the microservice utility is maximized without congestion, $i.e.$, $\mathcal{S}(\cdot) = 0$, and minimized when heavily congested $\mathcal{S}(\cdot) = 1$.
- d is the delay requirement of different microservices available at the ASAP microservice pool.

We assume that microservices are individually rational, $i.e.$, any microservice is attached to a specific application type, and it is characterized by a parameter $\mathcal{A}$. The microservice can choose between one of the two following conditions:

i) The available microservices is a critical microservice class; in this case, the designed utility function has to be the highest non-negative value, $\mathcal{K}$. Analytically:

$$\mathcal{K} = \arg\max_{i \in N} \mathcal{W}(\Theta, d; \mathcal{A}) \tag{2}$$

ii) To opt-out of the available microservices as background microservice, the microservice utility function is a negative value. Analytically:

$$\mathcal{W}(\Theta, d; \mathcal{A}) < 0, \forall i \in N \tag{3}$$

To combine the above two conditions, we consider another unique microservice class Υ for normal cloud microservices, which satisfies the condition $p_0 = 0$ and $d_0 = 1$. Therefore, any normal microservice can always choose microservice class Υ to gain zero utility. We denote the set of microservice types that choose the microservice class i as $\mathcal{A}_i(p; q)$. Notice that if the ASAP offers three microservice classes i, j, and k such that $p_i > p_j > p_k$ and $q_i > q_j > q_k$, then none of the microservices with class i will get priority in microservice provisioning, being inferior to class j and k in terms of both cost and delay. Here, p and q are defined as the priority classes based on the provisioning cost and delay. Without loss of generality, we hence can sort the indexes of non-dummy microservice classes in ascending order of priority level as follows:

$$p_1 > p_2 > \cdots > p_{\mathcal{N}} \tag{4}$$

$$q_1 > q_2 > \cdots > q_{\mathcal{N}} \tag{5}$$

Next, we discuss the resource provisioning scheme for microservices at the edge following the model.

B. Tradeoffs in Microservice Differentiation

In this section, we analyze the resource capacity constraints for different microservices. In particular, we focus on two typical scenarios, and they are discussed below:

Fixed Resource Provision. The normal microservices are supported by fixed resources, and hence the microservices with higher demand face a minimum resource capacity. Therefore, the normal microservices face difficulties while provisioning their computations at the edge. Analytically,

$$\mathcal{R}_i^n = \sigma p_i \sum_{t=1}^{T} \mathcal{F}_i^t \mathbb{R}_i^t \tag{6}$$

where σ denotes the scaling factor, p_i denotes the priority of microservice i, $\mathbb{R}_i^t$ denotes the fixed number of resource blocks allocated to microservice i at time t, $t \in T$, and $\mathcal{F}_i^t$ denotes the average microservice rate at time t.

Variable Resource Provision. The critical microservices are supported by variable resources, which are closer to the minimum requirements of the microservices. The variable microservices need to expand their network infrastructure to constantly support them. In this scenario, the microservices face the maximum resource capacity and also get the optimal amount of resource capacity invariably. We also provide extra resource blocks $\mathbb{R}_{extra}^t$ to the critical microservices if they require it in a specific period of time. We capture the resource capacity as $\mathcal{R}_c^i$, which is mathematically expressed:

$$\mathcal{R}_i^c = \sigma p_i \left(\mathbb{R}_{extra}^t + \theta \mathcal{F}_i^t \frac{\mathcal{R}_{req}}{\mathcal{R}_{tot}} \mathbb{R}_i^t \right) \tag{7}$$

where $\frac{\mathcal{R}_{req}}{\mathcal{R}_{tot}}$ denotes the microservice rate (*i.e.*, $\mathcal{R}_{req}$ and $\mathcal{R}_{tot}$ denote the required resources and total resource available for microservices.), θ denotes the scaling factor.

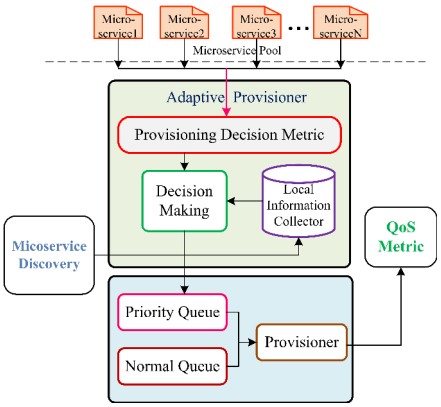


Figure 3: Architectural View of Microservice Provisioner

The microservice differentiator aims at finding the optimal microservice satisfying all resource capacity constraints to maximize the microservice utilization. The allocated resource provision mechanism needs to be maximized to find the optimal resource capacity in a fixed scenario. The microservice resource maximization problem is formulated.

$$\text{Max}\mathcal{H}_i = \begin{cases} \frac{\mathcal{R}_i^c}{\mathcal{R}_{th}^c}(\mathcal{V}_i - \mathcal{AS}_i(\Theta) - d_i), \text{critical microservice} \\ \frac{\mathcal{R}_i^n}{\mathcal{R}_{th}^n}(\mathcal{V}_i - \mathcal{AS}_i(\Theta) - d_i), \text{ Otherwise} \end{cases} \tag{8}$$

$$\text{subject to} \quad \mathcal{R}_i^c \geq \mathcal{R}_{th}^c, i \in N \tag{9}$$

$$\mathcal{R}_i^n \geq \mathcal{R}_{th}^n, i \in N \tag{10}$$

$$d_i \geq d_{th}, i \in N \tag{11}$$

$$\mathcal{S}_i(\Theta) \geq \mathcal{S}_{th}(\Theta) \tag{12}$$

where (8) describes the main function. (9) represents the critical resource constraint, $\mathcal{R}_i^c$, has to be greater than the threshold resource constraint, $\mathcal{R}_{th}^c$. The normal resource constraint, $\mathcal{R}_i^n$, has to be greater than the threshold normal resource constraint, $\mathcal{R}_{th}^n$, as shown in (10). (11) captures the microservice delay, d_i, that needs to be greater than the threshold microservice delay, d_{th}. Finally, the microservice utility discount function, $\mathcal{S}_i(\Theta)$, has to be greater than the threshold utility discount, $\mathcal{S}_{th}(\Theta)$, as shown in (12).

V. Adaptive Microservice Provisioning

Once the microservice differentiator has characterized the microservices, the microservice profile is sent to the microservice provisioner for resource provisioning; the aim is here to maintain better quality among all microservices. To provide such QoS, we design an adaptive and scalable microservice provisioner. We discuss the architecture and its mathematical model in the next subsections.

Near-Optimal Microservice Provisioner. The architecture of the Optimal Microservice Provisioner (OMP) is shown in Figure 3. In the microservice provisioner, all the microservices get buffered in a local microservice discovery module, and then they are transferred to the local information

collector module. Once collected by the decision-making module, the microservice provisioner starts the procedure described in Section V. Here, for the simplicity of the model, we have considered that the microservice belongs to a single application, whereas in a realistic scenario, one application may generate multiple microservices. With OMP, the microservices are guided into the provision engine module based on the provisioning decision metric in Equation (14) designed specifically for microservices. By considering the local microservice information by the microservice discovery module and provisioning decision metric, the provision engine module determines whether or not it needs to provision the microservices and to which server to provision. The data provisioner module maintains a critical priority queue and a normal queue, which efficiently provisions them to minimize delay and maintain fair QoS. The microservice provisioner is scalable, as it can process and provision different types of microservices efficiently.

Scalable Microservice Provisioning Scheme. Let us consider a system consisting of $\mathcal{J}$ edge servers. The prioritized microservices from the microservice differentiator need to be provisioned to these $\mathcal{J}$ edge servers during a large period $\{1, 2, \cdots, T\}$. The mobile devices must pay a cost to provision and execute the microservices within a specified time. The microservices arrive at the microservice provisioner according to their priority. Afterward, each microservice requests its resources to be mapped fairly within a virtual machine to provision and execute its processes. Before acquiring the resources to provision their computations, each microservice wields the following information $\mathcal{Y}_i^t$ at time t to the microservice provisioner:

i) R_i^h is the number of resources needed to provision the microservice i at time t, where h represents the total number of resource blocks required.
ii) $\mathcal{E}$ is the number of time slots required to provision the microservice i. However, the time slots allocated to microservices may not be consecutive. Instead of the time slots could be any time period to complete the microservice.
iii) $\mathcal{B}_i$ is the desired deadline to provision the microservice i in the edge server.
iv) $\mathcal{Q}_i$ is the penalty function designed to charge extra cost, if the specified deadline gets over for the particular microservice i.
v) $\mathcal{T}_i^{arri}$ and $\mathcal{T}_i^{com}$ denote the arrival and completion time of microservice i, respectively.

Our design also includes a penalty cost function. Such a penalty is imposed on all microservices that exceed their deadlines. The mathematical expression of the penalty function is given:

$$\mathcal{Q}_i = \begin{cases} \mathcal{C}_i^{pen}, & \text{if } (\mathcal{T}_i^{com} - \mathcal{T}_i^{arri}) < \mathcal{B}_i \\ \mathcal{C}_i^{pen} + \mathcal{X}, & \text{Otherwise} \end{cases} \tag{13}$$

where $\mathcal{C}_i^{pen}$ denotes the penalty cost if the microservice i exceeds its desired deadline and $\mathcal{X}$ denotes the penalty cost specific to a network operator. The penalty cost will be added to the total provisioning cost if microservice i exceeds a previously imposed deadline. If microservice i does not exceed its own desired deadline, the penalty cost will be deducted from the total provisioning cost. Mathematically, we have that,

$$\mathcal{C}_i^{tot} = \begin{cases} \mathcal{C}_i^{off} + \mathcal{C}_i^{pen}, & \text{if } (\mathcal{T}_i^{com} - \mathcal{T}_i^{arri}) < \mathcal{B}_i \\ \mathcal{C}_i^{off} - (\mathcal{C}_i^{pen} + \mathcal{X}), & \text{Otherwise} \end{cases} \tag{14}$$

where $\mathcal{C}_i^{off}$ denotes the provisioning cost of microservice i. The provisioning cost $\mathcal{C}_i^{off}$ of microservice i is expressed as:

$$\mathcal{C}_i^{off} = \sum_{j=1}^{\mathcal{J}} \sum_{t=1}^{T} r_i(t) \left(\mathcal{F}_i^{\mathcal{J}_j}(t) + \gamma_i(t) \frac{\theta_i^{\mathcal{E}}}{\mathcal{G}^{\mathcal{E}}} \right) \tag{15}$$

where $\gamma_i(t)$ denotes the unit resource allocation cost of microservice i at time t and $r_i(t)$ denotes the unit cost of microservice i provisioning at time t. $\theta_i^{\mathcal{E}}$ and $\mathcal{G}^{\mathcal{E}}$ denote the total resource blocks allocated to microservice i and total resource capacity, respectively. $\mathcal{F}_i^{\mathcal{J}_j}(t)$ denotes the cost of microservice i mapping to server j at time t. We describe the optimization problem for microservice provisioning cost $\mathcal{C}_i^{off}$, which is stated below. Here, we are trying to minimize the provisioning cost for edge devices to increase the network throughput. Mathematically,

$$\text{Min} \sum_{i=1}^{N} \mathcal{C}_i^{tot} = \sum_{i=1}^{N} \begin{cases} \mathcal{C}_i^{off} + \mathcal{C}_i^{pen}, \text{ if } (\mathcal{T}_i^{com} - \mathcal{T}_i^{arri}) < \mathcal{B}_i \\ \mathcal{C}_i^{off} - (\mathcal{C}_i^{pen} + \mathcal{X}), \text{ Otherwise} \end{cases} \tag{16}$$

$$\text{Subject to} \quad \sum_{i=1}^{N} \mathcal{C}_i^{off} \geq \mathcal{C}_{th}^{off}, i \in N \tag{17}$$

$$\sum_{i=1}^{N} \mathcal{C}_i^{pen} \geq \mathcal{C}_{th}^{pen}, i \in N \tag{18}$$

$$(\mathcal{T}_i^{com} - \mathcal{T}_i^{arri}) < \mathcal{B}_i, t \in \mathcal{T} \tag{19}$$

where Equation (16) describes the main objective function. Equation (17) represents that the total microservice provisioning cost, $\mathcal{C}_i^{off}$, is to be greater than the threshold microservice provisioning cost, $\mathcal{C}_{th}^{off}$. The microservice penalty cost, $\mathcal{C}_i^{pen}$, has to be greater than the threshold microservice penalty cost, $\mathcal{C}_{th}^{pen}$, as shown in Equation (18). Equation (19) captures the prescribed microservice deadline $\mathcal{B}_i$ required to be greater than the threshold $(\mathcal{T}_i^{com} - \mathcal{T}_i^{arri})$. After solving this optimization problem, all critical microservices will be provisioned immediately to edge servers. All other microservices (*i.e.*, microservice class 2 and 3) will be subsequently provisioned to cloud servers. According to Formulas (16), (17), (18), and (19), the provisioning mechanism is a nonlinear integer programming problem, and it

essentially turns out to be a combinational optimization of microservice provisioning with provisioning cost constraints and resource maximization with delay constraints. We solve the problem using Lagrangian Relaxation Theory [19].

VI. Performance Analysis

We present the evaluation results of ASAP with different experimental scenarios. We provide a detailed large-scale performance results using *CloudSim* [20] simulations and also provide test-bed experiments for microservices.

Table I: Experimental Settings

Parameter	Value
CPU bandwidth capacity	20 MHz
Total available microservice tasks	200
Microservice execution time	[5, 15] ms
CPU resources	[10, 20] MHz
Power consumption of mobile device (Tx)	100 mW
Processing capacity of mobile device	0.7 GHz
Processing capacity of MEC server	100 GHz
Microservice traffic arrival rate	[0, 10] unit/sec
Average microservice traffic rate	100 Mbits
SLAs for microservices (deadline)	[0, 10]

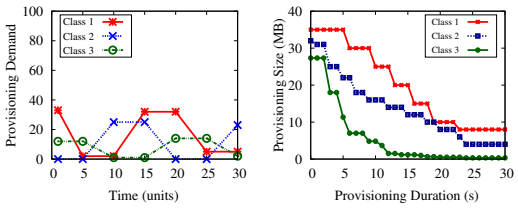
A. Evaluation Settings

We define the evaluation settings to depict the performance evaluation of ASAP and present all the parameters in Table I. We define several parameters as described in Table II and in line with existing literature [16], [17] to obtain realistic values regarding microservice configurations. For the edge computing platform, we have around 250 mobile devices dispersed over a populated area of 5 km X 5 km, and 3 powerful 5G-enabled base stations are located near MEC servers. The computational processing capability of MEC servers can be set to 100 GHz, and the computation processing capability of a mobile device is 0.95 GHz. The delay factor for the backhaul network is set to 0.0008 sec/KB [21]. To provision the microservices, we define the maximum time period distributed over 8-12 ms. The microservice size can be set from a range of 750-1250 KB.

Table II: A summary of microservices types and SLAs

Classes	Microservice Types	SLA
Class 1	Media Service	High
Class 2	Social Network	Moderate
Class 3	Hotel Reservation	Normal

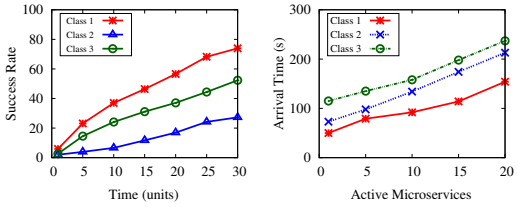
Prototype. We design a small setup comprised of 15 edge servers, the configuration of each server is as follows; Intel core-i9, 8 cores, 16 threads, operates at 3.6 GHz, 5 GHz maximum frequency, and 128 GB of memory with DDR4-2666 memory. These servers are operated on Ubuntu 14.04, 64-bit with Linux machine with kernel version 3.13.X, and out of these 15 edge servers, 10 are arranged for microservices and the other 5 for monolithic services. Basically, 10 of them are specifically designed for microservices with limited computational and resource capacity, whereas the other 5 of them are designed for monolithic services with higher computational and resource capacity. We design a client and server module for ASAP. The client generates the microservice traffic from edge applications and collects the information of SLAs for microservices at the application layer. The server module is responsible for the execution of tasks generated from different microservices on edge servers and the provision of the resources to them. The servers are enabled with Broadcom 43224AGN Gigabit Ethernet NICs, and they can be connected to each other with an ethernet switch with 144M pps and a bandwidth of 9Gbps. Such switches can hold up to 8 priority classes for queues.



(a) Demand Profile (b) Data Size

Figure 4: Analysis of provisioning demand and size.

Metrics. For microservice identification, we consider several performance metrics to quantify the microservice utilization and ASAP's accuracy (i.e., success rate). The success rate can be formulated as the ratio of the maximum number of microservice efficiently differentiated and the maximum number of microservices that participated in the process. As per the provision of microservices is concerned, we measure the resource utilization for microservices.



(a) Success Rate (b) Arrival Time

Figure 5: Analysis of success rate and arrival time.

Baselines. We consider two relevant solutions as baselines for the comparison with ASAP. MODEM is proposed by Fadda *et al.* [22]; this paper designed a quality-of-monitoring aware microservice deployment scheme. It designed a multi-objective linear optimization problem to find an optimal deployment configuration to maximize the quality of monitored data and minimize the overall cost. DEMIC is proposed by Wang [23]; this paper designed a delay-aware microservice coordination scheme for edge computing platforms. It uses online reinforcement learning to achieve minimal delay and migration cost.

B. Small-Scale Experiments

Impact on Provisioning Demand. Figure 4(a) represents the provisioning demand of different microservices for the edge platform. From Figure, we observe that microservice class 1 provides higher provisioning demand than microservice classes 2 and 3. Moreover, we observe how the provisioning demand for microservice class 1 increases as microservice class 1 provides real-time critical microservices. This is in line with recent applications of edge computing, e.g., IoT, in which critical microservice have higher demand. Hence, Figure 4(a) shows that the microservice class provides 7% higher provisioning demand compared to microservice classes 2 and 3. Also, microservice class 1 has 24% higher provisioning demand over microservice classes 2 and 3. Figure 4(b) shows the total data size provisioned to the edge platform for a fixed duration of time. We observe from our simulations that the provisioned data size decreases with the increase in the provision duration. As the data provisioning duration increases, the arrival rate of microservices at ASAP also increases [‡]. We explain this phenomenon with the increase of congestion in the network, which inherently increases the packet loss rate. From Figure 4(b), we can see how the provisioned data size increases for microservice class 1 than for microservice classes 2 and 3. As the proposed ASAP scheme efficiently provisioned the critical microservices to the edge platform while giving the highest priority. Due to the given priority, microservice class 1 faces a lower level of congestion in the network than other classes. Therefore, the provisioned data size also increases for microservice class 1. From Figure 4(b), we observe that microservice class 1 provides 14 and 22 $\%$ improvement compared to microservice classes 2 and 3, respectively.

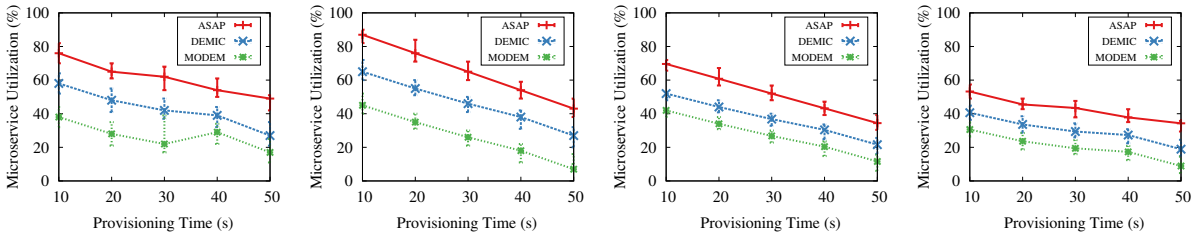
Impact on Success Rate. In Figure 5(a), we show the success rate of microservice differentiation for ASAP. From this simulation, we observe that the success rate of identifying the proper microservice classes increases with the total time. We analyzed such a success rate for 30 units of time. The success rate increases as ASAP efficiency chooses the different microservice classes. This is because ASAP estimates the microservice utility of different heterogeneous microservice available and takes an optimal decision to provide fair resources to each microservice. We also analyze the success rate for different microservice classes; microservice class 1 has a higher priority and provides a higher success rate. The success rate of microservice class 1 is higher than the microservice classes 2 and 3 by 15 and 22%, respectively. Figure 5(b) shows the arrival time of different microservices classes for ASAP. From the figure, we see that the arrival time of microservice class 1 is comparatively lesser than the microservice classes 2 and 3, respectively. The proposed microservice differentiator efficiently classifies the different microservice classes, and the highest priority is given. Therefore, the microservice class 1 is sent to the microservice provisioned while following the FIFO mechanism, which inherently decreases the arrival time. On the other hand, microservice classes 2 and 3 are given moderate and lower priority, respectively. Hence, microservice class 2 has a lower arrival time than microservice class 3.

[‡]As the arrival rate of microservices is higher than the departure rate.

C. Large-Scale Experiments

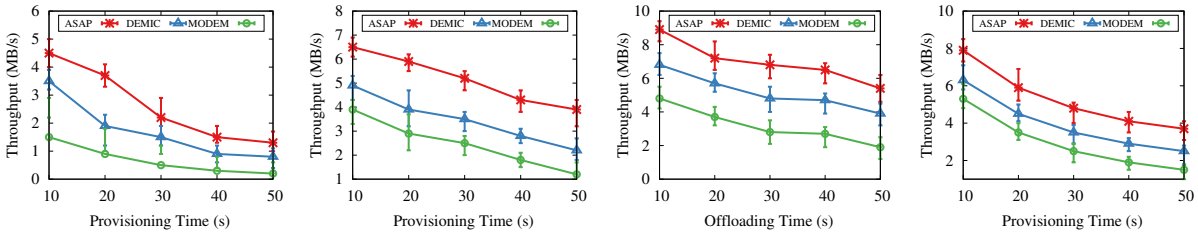
Impact on Microservice Utilization. Figure 6 shows the microservice utilization of ASAP. This parameter captures the scalability of ASAP. A schema with higher microservice utilization is a more scalable schema. From the figure, we observe that the microservice utilization decreases as the provision time increases in the network. We perform the experiments of ASAP with other two existing schemes DEMIC and MODEM; while varying the data provisioning rate $5MB/s$ and $10MB/s$. From figures 6(a) and 6(b), we found that the microservice utilization decreases with the increase in provision time. This is because as the provision time decreases, the packet loss rate increases, which inherently increases the microservice delay and microservice provisioning rate. It outperforms DEMIC and MODEM by 12-16 $\%$ for the data provisioning rate $5MB/s$ and $10MB/s$, respectively. We compared ASAP with the existing scheme DEMIC and MODEM while varying the load fraction 0.5 and 0.9. From figures 6(c) and 6(d), we observe that the ASAP's microservice utilization increases than the existing scheme – DEMIC and MODEM, as ASAP first accurately differentiate the proposer microservice class and later provisioned the microservices classes with different priority, which increases the efficiency of the microservice class 1. Thus, it outperforms DEMIC and MODEM by 8-10 $\%$ for varying load fractions 0.5 and 0.9.

Impact on Throughput. Figure 7 shows the throughput analysis of ASAP. From figures 7(a) and 7(b), we observe that the throughput decreases as the provision time increases. We perform the experiments in both the edge and cloud platforms while varying the data provisioning rate $5MB/s$ and $10MB/s$. From the figures, we also note that the network throughput decreases with the increase in provision time. This is because as the provision time decreases, the packet loss rate increases, which inherently increases the microservice delay and microservice provisioning rate. ASAP outperforms DEMIC and MODEM by 10-12 $\%$ for the data provisioning rate $5MB/s$ and $10MB/s$, respectively. We also compared ASAP with the existing approaches DEMIC and MODEM with the variation in load fraction 0.5 and 0.9. From figures 7(c) and 7(d), we observe that the ASAP's throughput increases than the naive approach, as ASAP first accurately differentiate the proposer microservice class and later provisions the microservices classes with different priority, which increases the efficiency of the microservice class 1. Hence, ASAP outperforms DEMIC and MODEM by 7-9 $\%$ for varying load fraction 0.5 and 0.9.



(a) Data Rate = 5 MB/s (b) Data Rate = 10 MB/s (c) Load Fraction = 0.5 (d) Load Fraction = 0.9

Figure 6: Analysis of microservice utilization with varying data rate and load fraction (Simulation)



(a) Data Rate = 5 MB/s (b) Data Rate = 10 MB/s (c) Load Fraction = 0.5 (d) Load Fraction = 0.9

Figure 7: Analysis of network throughput with varying data rate and load fraction (Simulation)

VII. Conclusion

In this paper, we presented ASAP, a novel architecture for edge platforms that adaptively identifies the optimal edge services to provision from a pool of heterogeneous applications. First, we model the problem of adapting service provisioning with optimization theory. Then we show with a prototype evaluation and extensive simulations that ASAP achieves over 20 % service type identification accuracy. Its throughput and delay performance outperform standard edge and cloud-based provisioning services. We also showed how ASAP increases the network throughput and fairness.

References

[1] A. Samanta and Z. Chang, "Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint," *IEEE IoT-J*, vol. 6, no. 2, pp. 3864–3872, 2019.

[2] L. Tong, Y. Li, and W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing," in *IEEE INFOCOM*, 2016, pp. 1–9.

[3] A. Samanta *et al.*, "Incentivizing microservices for online resource sharing in edge clouds," in *IEEE ICDCS*, 2019, pp. 420–430.

[4] A. Samanta, F. Esposito, and T. G. Nguyen, "Fault-tolerant mechanism for edge-based iot networks with demand uncertainty," *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 16 963–16 971, 2021.

[5] S. Deng *et al.*, "Composition-driven iot service provisioning in distributed edges," *IEEE Access*, vol. 6, pp. 54 258–54 269, 2018.

[6] X. Ma *et al.*, "Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing," *IEEE TCC*, 2019.

[7] Z. Zhou *et al.*, "Energy-efficient edge computing service provisioning for vehicular networks: A consensus admm approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5087–5099, 2019.

[8] X. Cao *et al.*, "Edge federation: Towards an integrated service provisioning model," *IEEE/ACM Transactions on Networking*, 2020.

[9] L. Zhang and N. Ansari, "Latency-aware iot service provisioning in uav-aided mobile edge computing networks," *IEEE Internet of Things Journal*, 2020.

[10] S. Gu, X. Luo, D. Guo, B. Ren, G. Tang, J. Xie, and Y. Sun, "Joint chain-based service provisioning and request scheduling for blockchain-powered edge computing," *IEEE IoT-j*, pp. 1–1, 2020.

[11] L. Li *et al.*, "Data-driven optimization for cooperative edge service provisioning with demand uncertainty," *IEEE IoT-J*, pp. 1–1, 2020.

[12] I. Filip *et al.*, "Microservices scheduling model over heterogeneous cloud-edge environments as support for iot applications," *IEEE IoT-J*, vol. 5, no. 4, pp. 2672–2681, 2018.

[13] H. Zhao *et al.*, "Distributed redundancy scheduling for microservice-based applications at the edge," *IEEE TSC*, pp. 1–1, 2020.

[14] L. Chen, Y. Xu, Z. Lu, J. Wu, K. Gai, P. C. Hung, and M. Qiu, "Iot microservice deployment in edge-cloud hybrid environment using reinforcement learning," *IEEE IoT-J*, pp. 1–1, 2020.

[15] Y. Wang *et al.*, "Mpcsm: Microservice placement for edge-cloud collaborative smart manufacturing," *IEEE TII*, pp. 1–1, 2020.

[16] A. Samanta and J. Tang, "Dyme: Dynamic microservice scheduling in edge computing enabled iot," *IEEE IoT-J*, vol. 7, no. 7, pp. 6164–6174, 2020.

[17] A. Samanta *et al.*, "Battle of microservices: Towards latency-optimal heuristic scheduling for edge computing," in *IEEE NetSoft*, 2019.

[18] A. Samanta, T. T. Ha, and T. G. Nguyen, "Distributed resource distribution and offloading for resource-agnostic microservices in industrial iot," *IEEE TVT*, pp. 1–12, 2022.

[19] O. Du Merle *et al.*, "A Lagrangian Relaxation of the Capacitated Multi-item Lot Sizing Problem Solved with an Interior Point Cutting Plane Algorithm," in *Approximation and Complexity in Numerical Optimization*. Springer, 2000, pp. 380–405.

[20] R. N. Calheiros *et al.*, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Soft.: Practice and exp.*, vol. 41, no. 1, pp. 23–50, 2011.

[21] K. Zhang *et al.*, "Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[22] E. Fadda, P. Plebani, and M. Vitali, "Monitoring-aware optimal deployment for applications based on microservices," *IEEE TSC*, pp. 1–1, 2019.

[23] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. S. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE TMC*, pp. 1–1, 2019.