







Incremental Multilayer Resource Partitioning for Application Placement in Dynamic Fog

Zahra Najafabadi Samani , Narges Mehran , Dragi Kimovski , Shajulin Benedict ,
Nishant Saurabh , and Radu Prodan 

Abstract—Fog computing platforms became essential for deploying low-latency applications at the network’s edge. However, placing and managing time-critical applications over a Fog infrastructure with many heterogeneous and resource-constrained devices over a dynamic network is challenging. This paper proposes an incremental multilayer resource-aware partitioning (M-RAP) method that minimizes resource wastage and maximizes service placement and deadline satisfaction in a dynamic Fog with many application requests. M-RAP represents the heterogeneous Fog resources as a multilayer graph, partitions it based on the network structure and resource types, and constantly updates it upon dynamic changes in the underlying Fog infrastructure. Finally, it identifies the device partitions for placing the application services according to their resource requirements, which must overlap in the same low-latency network partition. We evaluated M-RAP through extensive simulation and two applications executed on a real testbed. The results show that M-RAP can place 1.6 times as many services, satisfy deadlines for 43% more applications, lower their response time by up to 58%, and reduce resource wastage by up to 54% compared to three state-of-the-art methods.

Index Terms—Application placement, deadline satisfaction, Fog computing, resource partitioning, resource wastage.

I. INTRODUCTION

HIGHLY distributed applications, such as video processing [1], e-commerce [2], virtual reality [3], object recognition, face detection [4], or natural language processing [5], raise significant time-critical and low-latency challenges impossible to fulfill using centralized Cloud data center technologies [6]. To mitigate this problem, *Fog computing* [7] has emerged as a distributed paradigm encompassing highly heterogeneous and dynamic devices, hierarchically split between the high-end

Manuscript received 3 March 2022; revised 5 March 2023; accepted 25 March 2023. Date of publication 28 March 2023; date of current version 8 May 2023. This work was supported by the European Horizon 2020 project DataCloud under Grant 101016835, in part by the Horizon Europe project Graph-Massivizer under Grant 101093202, and in part by the Austrian Research Promotion Agency (FFG) project Kärntner Fog under Grant 888098. *Recommended for acceptance by S. Pallickara. (Corresponding author: Radu Prodan.)*

Zahra Najafabadi Samani, Narges Mehran, Dragi Kimovski, and Radu Prodan are with the Institute of Information Technology, University of Klagenfurt, 9020 Klagenfurt, Austria (e-mail: Zahra.Najafabadi@aau.at; narges.mehran@aau.at; dragi.kimovski@aau.at; radu.prodan@aau.at).

Nishant Saurabh is with the Department of Information and Computing Sciences, Utrecht University, 3584 CS Utrecht, The Netherlands (e-mail: n.saurabh@uu.nl).

Shajulin Benedict is with the Computer Science and Engineering, Indian Institute of Information Technology Kottayam, Valavoor, Kerala 686635, India (e-mail: shajulin@iiitkottayam.ac.in).

Digital Object Identifier 10.1109/TPDS.2023.3262695

TABLE I
MOTIVATIONAL SERVICE PLACEMENT EXAMPLE

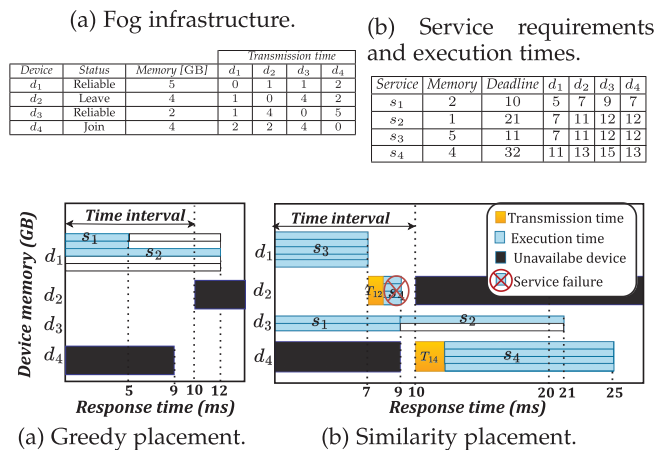


Fig. 1. Comparative placement Gantt charts for Example 1.

Cloud and the low-end user devices. The Fog supports applications with time-critical low-latency constraints through service placement [6], [8] across devices closer to the users’ needs (e.g., computation, communication, storage). Existing works use resource partitioning [9], machine learning [10], or distributed deadline-aware [11] methods to place time-critical applications in the Fog. Despite employing different technologies, they all use greedy heuristics that overload the fastest devices to lower the response time. However, Fog infrastructures have limited capacity-constrained resources with sporadic availability, requiring a trade-off between utilization and response time to satisfy various time-critical application requests. The following example illustrates the disadvantage of greedy heuristics mitigated by a *similarity placement* method.

Example 1. Let us assume a Fog network of four devices with different memory sizes (see Table I). The devices d_1 and d_3 are always available. We assume two concurrent applications $A_1 = \{s_1, s_2\}$ and $A_2 = \{s_3, s_4\}$ of two sequential services each, with specific memory and execution deadline requirements (see Table I b). The services exchange a message of fixed size, transmitted between the hosting devices with the duration shown in Table I a. After 9ms from the application requests, device d_2 joins and d_4 leaves the network. To mitigate the failures caused by the leaving device d_4 , we consider a monitoring interval of 10ms, equal to the minimum service deadline. Fig. 1 displays the Gantt charts comparing two placement heuristics.

- 1) *Greedy placement* unnecessarily overloads the fastest device d_1 , leading to *resource wastage* (representing the percentage of unused memory) and deadline violations. For example, placing s_1 onto d_3 satisfies its required memory and provides a response time of 9ms ahead of its deadline. However, the method places s_1 with the lowest deadline onto d_1 and occupies 40% of its memory with a low response time of 5ms. Similarly, it places s_2 with a lower deadline on d_1 , eliminating the transmission time and providing the lowest response time of $5 + 0 + 7 = 12ms$. The utilization of d_1 is 60%, and the remaining devices have insufficient memory to host s_3 despite being idle. Moreover, they cannot host s_4 due to the dependency on s_3 . The method places half of the services and, thus, has a low *placement rate* of 0.5 and a *deadline satisfaction rate* of 0.5. The average device memory wastage is $1 - \frac{2+1}{15} \approx 0.8$ for the first monitoring interval and $1 - \frac{1}{11} \approx 0.9$ for the second.
- 2) *Similarity placement* maps each service to a device with a capacity similar to the required resources and a response time within the deadline. Unlike the greedy method, it places s_1 to d_3 with an equal memory size and a response time of 9ms, eliminating the wastage on d_3 , satisfies the deadline, and spares d_1 for other services. Likewise, placing s_2 to d_3 eliminates the transmission time and generates a response time of $9 + 0 + 12 = 21ms$, which satisfies the deadline. Afterward, it places s_3 onto d_1 with a response time of 7ms, which satisfies its deadline and eliminates memory wastage. Finally, it places s_4 on all 4GB of memory of d_2 with a lower transmission to d_1 . As d_2 leaves the network before completing s_4 , the placement relocates s_4 onto d_4 with a failure overhead of 3ms (i.e., 2ms execution on the failed device plus 1ms to detect the failure), a transmission time of 2ms (from d_1 , hosting s_3) and a response time of $7 + 3 + 2 + 13 = 25ms$. Despite the lost computation of 3ms, s_4 can still reach its deadline. The average resource wastage is $1 - \frac{5+4+2}{15} \approx 0.27$ for the first interval, and $1 - \frac{0+1+4}{11} \approx 0.55$ for the second. Notably, although a lower resource wastage leads to a higher placement rate, one can still reach a high placement rate and waste resources.

Research Proposal. To address this challenge, we propose an *incremental multilayer resource-aware partitioning (M-RAP)* method for adaptive placement of *distributed time-critical applications*, represented as directed acyclic graphs (DAG) of services with *soft absolute response deadlines* [12], in a dynamic Fog. M-RAP approaches this problem as multi-objective optimization of three goals: *Fog placement rate*, *resource wastage*, and *service deadline satisfaction rate*. M-RAP models Fog as a dynamic *multilayer graph* comprising the changing network structure [13] and devices with different resources and availability [14]. M-RAP splits the Fog devices into overlapping partitions considering different resources and network structures to accelerate application placement. Afterward, it incrementally updates the partitions by tracking infrastructure changes to detect clusters of dynamic networks with minimum time and cost that avoid re-partitioning. Each resource partition has an

associated *feature*, defined as a quadruplet of the average number of cores, memory and storage sizes, and processing speed of its devices. Afterward, the dynamic application placement needs two steps.

- 1) *Similarity placement* step maps all application services to feature partitions based on their resource requirements, which must share the same network partition underneath to satisfy their deadlines.
- 2) *Monitoring and replacement* step mitigates critical situations in case of device failures and invalid placements and maps the affected services onto new feature partitions. We performed extensive simulations with an intensive workload of application requests in a Fog environment with devices that dynamically join or leave the network. Compared to three related methods [9], [15], [16], M-RAP improves the Fog placement rate by range 1.4–1.6 times, satisfies deadlines for range 5–43% more placement requests, decreases the application response time by range 16–58%, and reduces the resource wastage by range 1–54%. We validate the simulation using two e-commerce and video streaming applications in a real computing continuum testbed [17].

Extensions. This paper extends our early work [18] on application placement in a static multilayer Fog in three areas.

- 1) We model the evolving Fog as a dynamic multilayer graph based on the incremental network changes and device availability;
- 2) We design a new dynamic placement algorithm based on an incremental multilayer resource partitioning method considering infrastructure changes;
- 3) We compare our results against three state-of-the-art methods using two applications running on a real testbed.

Outline. The paper has ten sections. Section II summarizes the related work. Section III presents the model underneath M-RAP representing the Fog as a dynamic multilayer graph. Section IV proposes a two-phase architecture for placing an application in a dynamic Fog: multilayer partitioning described in Section V and service placement described in Section VI. Section VII provides the experimental setup. Section VIII evaluates M-RAP compared to related work using simulation experiments, confirmed on a real testbed in Section IX. Section X concludes the paper.

II. RELATED WORK

This section revisits existing static and dynamic Fog placement methods, summarized in Table II.

A. Static Placement

Static placement methods model the Fog without inspecting the unstable network and the dynamic service workloads.

- 1) *Resource wastage:* Taneja et al. [15] proposed a resource-aware application mapping to maximize utilization in Fog. Similarly, Shooshtarian et al. [19] proposed a two-phase allocation method on hierarchical Fog resources using local clustering in each layer to optimize resource utilization. Jie et al. [20] modeled resource allocation in Fog and Cloud as a two-stage non-cooperative game to minimize the service cost and maximize

TABLE II
RELATED WORK COMPARISON (“D”: DEADLINE SATISFACTION, “T”: MESSAGE TRANSMISSION, “W”: RESOURCE WASTAGE)

Related Work	Objective			Application		Scheduling Problem	Infra-structure	Placement Algorithm	Simulation Tool	Real Testbed
	D	T	W	Model	Requirement					
[15]	–	–	✓	DAG	static	uniform	static	Binary search	iFogSim	–
[19]	–	–	✓	Bag	static	uniform	static	Graph theory	Matlab	–
[20]	✓	–	✓	Bag	static	uniform	static	Game theory	Custom	–
[21]	–	✓	–	Bag	static	uniform	static	Graph theory	Custom	–
[22]	–	✓	–	Bag	static	uniform	static	Graph theory	CloudSim	–
[23]	–	✓	–	Bag	static	uniform	static	NSGA-II	Matlab	–
[9]	–	✓	–	DAG	static	uniform	static	ILP/Graph theory	Yafs	–
[24]	–	✓	–	Bag	static	uniform	static	ILP	Custom	–
[25]	✓	–	✓	Bag	dynamic	uniform	static	Greedy	FogPlan	–
[10]	✓	–	–	Bag	dynamic	uniform	static	Machine learning	Custom	–
[11]	✓	✓	–	Bag	dynamic	unrelated	static	Greedy	–	✓
[26]	–	✓	✓	Bag	dynamic	unrelated	static	Greedy	Custom	–
[27]	–	✓	–	Bag	dynamic	unrelated	static	ILP	Custom	–
[28]	✓	–	✓	Bag	static	uniform	dynamic	Particle Swarm	Custom	–
[16]	✓	–	✓	Bag	static	uniform	dynamic	Particle Swarm	Custom	–
M-RAP	✓	✓	✓	DAG	static	uniform	dynamic	Graph theory	Yafs	C ³

resource utilization. Unfortunately, these methods ignore the network structure of Fog devices and introduce high latencies.

2) *Transmission time*: Sajjad et al. [21] proposed a decentralized resource partitioning method based on multiple biased random walks to reduce the transmission time. Similarly, Filiposka et al. [22] designed a community-based resource management method in Fog that exploits distributed hierarchical clustering to minimize latency and service migration. Sun et al. [23] partitioned the Fog devices based on the network connection and modeled the application placement in each cluster as a multi-objective optimization of execution and transmission time. Lera et al. [9] proposed a greedy application placement that optimizes availability and latency by partitioning the Fog resources based on their connectivity. Ma et al. [24] introduced a collaborative method for service caching and placement in edge computing formulated as a mixed integer nonlinear programming problem to minimize the transmission and execution time. These methods clustered the Fog devices based on network connectivity without considering the other resources such as processing speed, memory, or storage.

3) *Contribution*: Static placements methods investigate greedy heuristics to improve transmission time [9], [21], [22], [23], [24] by overloading the fast devices and generating high wastage. Although [15], [19], [20] improve resource wastage, they impose high latencies. M-RAP improves these methods by clustering the Fog devices into overlapping partitions based on their network structure and resources optimizing transmission time and resource wastage.

B. Dynamic Placement

Related work studied two dynamic placement aspects.

1) *Dynamic applications*: Yousefpour et al. [25] investigated changes in the application structure by formulating the placement as an integer non-linear programming problem using reactive service provisioning for optimizing cost and service delay. Similarly, Sami et al. [10] inspected the changes in the application structure using a proactive demand-driven deep reinforcement method to optimize the deadline. Meng et al. [11] proposed a deadline-aware task dispatching and scheduling method for Edge computing as an unrelated parallel machine problem that minimizes resource wastage and transmission

TABLE III
IMPORTANT NOTATION SUMMARY

	Notation	Definition
Infrastructure	d_j^t	Fog device during time interval Δt
	R_{jk}^t	available resource k on device d_j^t during time interval Δt
	LAT_{ij}^t/BW_{ij}^t	latency / bandwidth between devices d_i^t and d_j^t during time interval Δt
	G_t	Fog multilayer graph during time interval Δt
	E_{il}/E_{ii}	inter-layer / intra-layer edges
	l_0, l_1, l_2, l_3	network, processing, memory, and storage layer
	ΔG_t	multilayer graph changes during time interval Δt
	GJ_t/GL_t	incremental growth / shrink
	DJ_t/DL_t	joining / leaving devices during time interval Δt
	Application	A
s_i		application service
m_{pi}		request message of service s_p to service s_i
SZ_{pi}		size of message of m_{pi}
$r_{i1}, r_{i2}, r_{i3}, r_{i4}$		number of cores, memory, storage, workload of service s_i
θ_i/θ		service s_i / application deadline
u		user requesting application A
Place.	TR_{ij}^t	transmission time between devices d_i^t and d_j^t
	ET_{ij}^t/RT_{ij}^t	execution / response time of service s_i on device d_j^t during time interval Δt
	RT_A	response time of application A
Partitioning	Q	modularity
	$\mathcal{P}_t(l)/\mathcal{P}_t(G_p)$	layer / feature partition during time interval Δt
	FP_k^t	feature partition k in $\mathcal{P}_t(G_p)$ during time interval Δt
	G_p^t	compressed graph at during time interval Δt
	$\Delta \mathcal{P}_t(l)/\Delta G_p^t$	layer partition / compressed graph changes during time interval Δt

time through a decentralized greedy heuristic. Similarly, Han et al. [26] formulated a linear programming scheduling problem to minimize the response time of Edge computing services by considering the dynamic application requirements in a static Fog network. Tran et al. [27] proposed a collaborative caching and processing based on an integer linear programming Edge scheduling method to minimize the transmission time considering dynamic application requests.

2) *Dynamic infrastructure*: Mseddi et al. [28] modeled the availability of Fog devices based on discrete Markov processes to maximize the application deadline satisfaction. They extended their work with an integer linear programming method [16] to optimize the monolithic application placement rate and deadline satisfaction, considering the mobility of Fog devices rather than workflows.

3) *Contribution*: Most related works [10], [11], [24], [25], [26] focused on dynamic workflows and neglected network instability that affects their deadlines. A few efforts [16], [28] partially considered the deadlines or resource wastage in a dynamic Fog for simpler monolithic applications. M-RAP augments the related works by modeling the network instability and device availability as a dynamic multilayer graph to decrease resource wastage and maximize deadline satisfaction of workflow applications in Fog.

III. MODEL

This section presents the abstract model underneath M-RAP, using a formal notation summarized in Table III.

A. Infrastructure Model

We model the distributed *infrastructure* in three layers.

- 1) *Cloud layer* represents a high-performance data center suitable for placing resource-intensive tasks.
- 2) *Fog network layer* $F = (D, N)$ lies between the Cloud and the end-users and provides proximity computational and storage services on top of two resource sets [7].

- *Physical devices* $D = \{d_1, d_2, \dots, d_n\}$, modeled as a quadruple of resources $d_j = (R_{j1}, R_{j2}, R_{j3}, R_{j4})$, where R_{j1} represents the number of cores, R_{j2} the memory size in GB, and R_{j3} the storage size in GB, and R_{j4} processing speed in millions of instructions (MI) per second.
 - *Network connections* $N = \{n_{qj} | (d_q, d_j) \in D \times D\}$ between devices, where a connection $n_{qj} = (BW_{qj}, LAT_{qj})$ depends on the bandwidth BW_{qj} and the latency LAT_{qj} between the devices d_q and d_j .
- 3) *Client layer* consists of a set of users $\mathcal{U} = \{u_1, \dots, u_z\}$, including sensor and actuator client devices that request Fog resources for placing their applications.

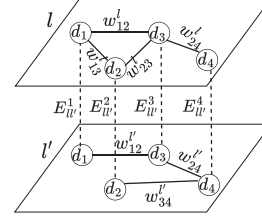


Fig. 2. Multilayer graph.

Example 2. Fig. 2 shows a weighted multilayer graph $G = (D, \mathcal{E}, L)$ with two layers $L = \{l, l'\}$ containing four devices $D = \{d_1, d_2, d_3, d_4\}$, and the intra-layer edges $E_{ll} = \{(w_{12}^l), (w_{13}^l), (w_{23}^l), (w_{24}^l)\}$ in the layer l , and $E_{l'l'} = \{(w_{12}^{l'}), (w_{24}^{l'}), (w_{34}^{l'})\}$ in the layer l' .

B. Multilayer Fog Resource Model

The single Fog network layer only represents the topological interconnection of the physical devices, thereby failing to capture their heterogeneous resources. To better handle resource diversity, we model the Fog across four layers $L = \{l_0, l_1, l_2, l_3\}$ that incorporates various device relationships based on the network topology and resource types.

- *Network layer* l_0 corresponds to the Fog network layer modeled in Section III-A.
- *Processing layer* l_1 indicates similarities among Fog devices according to their number of cores and their speed;
- *Memory* l_2 and *storage* l_3 layers indicate similarities based on Fog devices' memory and storage sizes.

We model *Fog resources* as a fully interconnected *multilayer graph* $G = (D, \mathcal{E}, L)$, where D is the set of Fog devices replicated across all four layers L , and $\mathcal{E} = \{E_{ll'} | \forall l, l' \in [0, L]\}$ is the set of weighted bidirectional graph edges of two types.

- 1) *Inter-layer edges* $E_{ll'} = \{E_{ll'}^l | (d_j, d_j) \in D \times D\}$ connect each device d_j in the layer $l \in L$ with the corresponding device d_j in all the other layers $l' \in L, l \neq l'$. They uncover relations among resource types within the same device. We consider a weight $w_{ll'}^j = 1$ for the inter-layer edges of the device d_j in the layers l and l' .
- 2) *Intra-layer edges* $E_{ll} = \{E_{ll}^l | (d_q, d_j) \in D \times D \wedge q \neq j\}$ connect two Fog devices inside one layer $l \in L$ using a weight function $w_{ll}^{(l)} : E_{ll} \rightarrow \mathbb{R}$, representing their *similarity score* $w_{qj}^{(l)} = \frac{1}{1 + \text{dis}_l(d_q, d_j)}$, where $\text{dis}_l(d_q, d_j)$ is the euclidean distance between their resources, calculated based on the features in each layer, as follows:

$$\text{dis}_l(d_q, d_j) = \begin{cases} \sqrt{(\hat{R}_{q1} - \hat{R}_{j1})^2 + (\hat{R}_{q4} - \hat{R}_{j4})^2}, & l = l_1; \\ R_{ql} - R_{jl}, & l \in \{l_2, l_3\}, \end{cases}$$

where \hat{R}_{j1} and \hat{R}_{j4} are the normalized number of cores R_{j1} and processing speed R_{j4} of d_j . This facilitates the partitioning of Fog devices based on their specific resource capacity. The Fog devices with a similarity score of 1 have an identical resource in the layer l .

C. Dynamic Fog Network Model

We define a *dynamic Fog infrastructure* as a temporal sequence of network layers $\{F_0, \dots, F_{t-1}, F_t, \dots\}$ changing over time. Every temporal Fog network layer $F_t = (D_t, N_t)$ during the time interval $\Delta t = [t-1, t]$ consists of:

- 1) *Physical devices* as unordered set $D_t = \{d_1^t, \dots, d_n^t\}$, where $d_j^t = (R_{j1}^t, R_{j2}^t, R_{j3}^t, R_{j4}^t)$ represents the resources available on a device d_j^t during the time interval Δt . We update the set of available devices $D_{t+1} = D_t \cup DJ_t \setminus DL_t$ with:
 - a) *Joining devices* DJ_t in the Fog network during the time interval Δt ;
 - b) *Leaving devices* DL_t from the Fog network during the time interval Δt .
- 2) *Network connections* $N_t = \{n_{qj}^t | (d_q^t, d_j^t) \in D_t \times D_t\}$ during the time interval Δt , where a network connection $n_{qj}^t = (BW_{qj}^t, LAT_{qj}^t)$ represents the available bandwidth BW_{qj}^t and latency LAT_{qj}^t between the devices d_q^t and d_j^t .

D. Dynamic Multilayer Fog Resource Model

We model the *dynamic Fog resources* as a temporal sequence of multilayer graphs $\{G_0, \dots, G_{t-1}, G_t, \dots\}$, where $G_t = (D_t, \mathcal{E}_t, L)$ models the the multilayer Fog resources during the time interval Δt (see Section III-B). We represent the *multilayer graph changes* $\Delta G_t = (GJ_t, GL_t)$ in the Fog infrastructure during the time interval $\Delta t = [t-1, t]$ using the incremental growth GJ_t and shrink GL_t .

- 1) *Incremental growth* $GJ_t = (DJ_t, \mathcal{E}J_t)$ denotes the set of joining devices $DJ_t = \{d_{n+1}, \dots, d_{n+k}\}$ and their associated edges $\mathcal{E}J_t = \{E_{ll'}^t | \forall l, l' \in [0, L]\}$ in the Fog network during the time interval Δt . We update the multilayer resource graph by replicating all new Fog devices in all the layers L of the multilayer graph G_t . We finally generate the set of bidirectional edges $\mathcal{E}J_t$ containing the new inter-layer edges $EJ_{ll'} = \{(d_j^t, d_j^t) \in DJ_t \times DJ_t\}$ and the new intra-layer edges $EJ_{ll} = \{E_{qj}^{lt} | (d_q^t, d_j^t) \in D_t \times DJ_t \wedge q \neq j\}$ associated to the Fog devices in DJ_t (see Section III-B).

- 2) *Incremental shrink* $GL_t = (DL_t, \mathcal{E}L_t)$ denotes the set of leaving devices $DL_t = \{d_{n-1}, \dots, d_{n-k'}\}$ and their associated edges $\mathcal{E}L_t = \{E_{ll'}^t | \forall l, l' \in [0, L]\}$ in the Fog network during the time interval Δt . We update the Fog multilayer graph by deleting the devices in DL_t and their associated inter-layer $EL_{ll'} = \{(d_j^t, d_j^t) \in DL_t \times DL_t\}$ and intra-layer edges $EL_{ll} = \{E_{qq}^t | (d_q^t, d_j^t) \in D_t \times DL_t \wedge q \neq j\}$ from the dynamic multilayer resource graph G_t .

Example 3. We consider the incremental changes in the multilayer graph $\Delta G_t = (GJ_t, GL_t)$ from Fig. 2 based on the incremental growth and shrink.

- 1) *Incremental growth* GJ_t considers the joining device $DJ_t = \{d_5\}$, its associated intra-layer edges $EJ_{ll} = \{w_{45}^t, w_{25}^t\}$ in the layer l , $EJ_{ll'} = \{w_{45}^t, w_{25}^t\}$ in the layer l' , and the inter-layer edge $EJ_{ll'} = \{E_{ll'}^5\}$.
- 2) *Incremental shrink* GL_t considers the leaving device $DL_t = \{d_1\}$, its associated intra-layer edges $EL_{ll} = \{w_{12}^t, w_{13}^t\}$ in the layer l , $EJ_{ll'} = \{w_{12}^t\}$ in the layer l' , and the inter-layer edge $EJ_{ll'} = \{E_{ll'}^1\}$.

E. Application Model

An *online time-critical application* has three characteristics.

- 1) Application structure $A = (\mathcal{S}, M, \Theta, u)$ requested by a user u is a DAG of *services* $\mathcal{S} = \{s_1, s_2, \dots, s_m\}$ interconnected through request *messages* M . Every service $s_i \in \mathcal{S}$ has a quadruple of resource demands $s_i = (r_{i1}, r_{i2}, r_{i3}, r_{i4})$, where r_{i1} is the required number of cores, r_{i2} is the required memory size, r_{i3} is the storage size, and r_{i4} is the workload in MI. We assume that the service resource demand is static and does not change over time.
- 2) Request message $m_{pi} = (SZ_{pi}, s_p, s_i) \in M$ has a size SZ_{pi} , a source $s_p \in \mathcal{S}$, and a destination service $s_i \in \mathcal{S}$. A user $u \in \mathcal{U}$ triggers the application execution via an *initial request message* m_{u1} to the service s_1 .
- 3) Absolute soft deadlines $\Theta = \{\theta_1, \theta_2, \dots, \theta_m\}$ define the target completion time θ_i for each service s_i . Following the theory of soft real-time computing, our utility function maximizes the set of met service deadlines to optimize the overall application quality of service [29].

F. Workload Model

We consider a *workload* $W = \{AS_0, \dots, AS_t, \dots\}$ of applications requested by different users \mathcal{U} during multiple time intervals $\Delta t = [t-1, t]$ in a dynamically evolving Fog infrastructure $\{F_0, \dots, F_t, \dots\}$.

- 1) *Placement* of an application $A = (\mathcal{S}, M, \Theta, u)$ during the time interval Δt is a function $\mu_t : \mathcal{S} \rightarrow D_t \cup \emptyset$, where $\mu_t(s_i) = d_j^t$ satisfies the needs of each service $s_i = (r_{i1}, r_{i2}, r_{i3}, r_{i4})$ on a device $d_j^t = (R_{j1}^t, R_{j2}^t, R_{j3}^t, R_{j4}^t)$: deadline satisfaction ($RT_{ij} \leq \theta_i$, defined in the next paragraphs), sufficient cores ($r_{i1} \leq R_{j1}^t$), memory ($r_{i2} \leq R_{j2}^t$) and storage ($r_{i3} \leq R_{j3}^t$).

An *invalid placement* $\mu_t(s_i) = \emptyset$ indicates no device in D_t that satisfies the service constraints.

- 2) *Execution time* of a service s_i is the ratio between its workload r_{i4} and the processing speed R_{j4} of the hosting device $d_j^t = \mu_t(s_i)$: $ET_{ij} = \frac{r_{i4}}{R_{j4}}$.
- 3) *Transmission time* of a message $m \in M$ of size SZ between two devices d_q^t and d_j^t is: $TR_{qj}^t = LAT_{qj}^t + \frac{SZ}{BW_{qj}^t}$, where LAT_{qj}^t is the latency and BW_{qj}^t is the bandwidth of a network connection $n_{qj}^t \in N_t$ at the time interval δt .
- 4) *Failure overhead* caused by a leaving device $d_j^t \in DL_t$ to a service s_i during the time interval Δt is the sum of its response time RT_{ij}^t and the time PT_{ij} for executing the placement algorithm to select a new device d_j^t (see Algorithm 3, Section VI-A): $FT_{ij}^t = RT_{ij}^t + PT_{ij}$.
- 5) *Response time* of a service $s_i \in \mathcal{S}$ running on the device $d_j^t = \mu_t(s_i)$ during the time interval Δt is the sum of
 - a) the maximum response time RT_{pq}^t of its predecessors s_p , including its request message transmission time TR_{qj}^t , where $\mu_t(s_p) = d_q^t$ (or TR_{uj}^t , in case of initial message request),
 - b) its execution time ET_{ij} , and
 - c) any potential failure overhead caused by the leaving device d_j^t :

$$RT_{ij}^t = \begin{cases} TR_{uj}^t + ET_{ij} + FT_{ij}^t, & \exists m_{ui} \in M; \\ \max_{m_{pi} \in M} \{RT_{pq}^t + TR_{qj}^t\} \\ + ET_{ij} + FT_{ij}^t, & \exists m_{pi} \in M \wedge s_p \in \mathcal{S}. \end{cases}$$

- 6) *Device allocation* resulting from a service placement $d_j^t = \mu_t(s_i)$ updates its resource availability $d_j^t = (R_{j1}^t - r_{i1}, R_{j2}^t - r_{i2}, R_{j3}^t - r_{i3})$ for the required execution time intervals: $t, t+1, \dots, t + \left\lceil \frac{RT_{ij}^t}{\Delta t} \right\rceil$, and releases them for the intervals following its completion.
- 7) *Application response time* is the highest response time of all services $s_i \in \mathcal{S}$: $RT_A = \max_{s_i \in \mathcal{S}} \{RT_{ij}^t | d_j^t = \mu_t(s_i)\}$. The service with the highest response time has no successors in the application DAG structure.
- 8) *Deadline fulfillment* requires that the response time of the application placement satisfies the deadline Θ : $RT_A < \Theta$.

G. Placement Objectives

We define three performance objectives for placing a requested application set AS_t in a dynamic Fog infrastructure F_t during the time interval $\Delta t = [t-1, t]$:

- 1) Maximize Fog placement rate, as the ratio between the number of services s_i placed in the Fog and the total number of requested services for all applications $A \in AS_t$: $\frac{\sum_{A \in AS_t} |\mathcal{S}_\mu|}{\sum_{A \in AS_t} |\mathcal{S}|}$, where $\mathcal{S}_\mu = \{s_i \in \mathcal{S} | \mu_t(s_i) \neq \emptyset\}$ and $|\mathcal{S}|$ and $|\mathcal{S}_\mu|$ represent the cardinality of the two service sets.
- 2) Minimize processing, memory, and storage wastages, defined as the remaining percentage between the resource units r_{ik} consumed by the placed services to the total resource units R_{jk}^t of the devices: $1 - \frac{\sum_{A \in AS_t} \sum_{s_i \in \mathcal{S}_\mu} \max\{\frac{r_{ik}}{unit_k}\}}{\sum_{d_j^t \in D_t} \max\{\frac{R_{jk}^t}{unit_k}\}}$, where $k \in 1, 2, 3$, $unit_1 =$

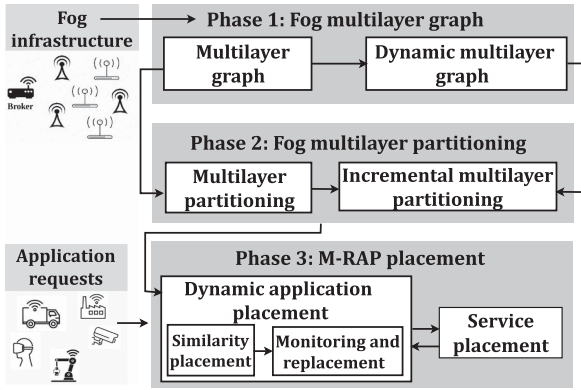


Fig. 3. M-RAP architecture design.

1core and $unit_2 = unit_3 = 1\text{GB}$ (memory and storage sizes).

- 3) Maximize service deadline satisfaction rate, defined as the percentage of services fulfilling their absolute soft deadlines: $\frac{\sum_{A \in AS_t} |\mathcal{S}_\Theta|}{\sum_{A \in AS_t} |S|}$, where $\mathcal{S}_\Theta = \{s_i \in S \mid RT_{ij}^t \leq \theta_i\}$, and $|\mathcal{S}_\Theta|$ along with $|S|$ represent the cardinality of the two service sets.

IV. M-RAP ARCHITECTURE DESIGN

Fig. 3 depicts the M-RAP architecture design in three phases, outlined in this section: multilayer graph modeling, Fog multilayer partitioning, and M-RAP placement.

A. Multilayer Graph Modeling

A *multilayer graph* models the Fog devices based on their network interconnections and similarity in processing speed and the number of cores, memory, and storage sizes, as described in Section III-B. A *dynamic multilayer graph* constantly updates the multilayer graph based on the incremental temporal changes (i.e., growth, shrink) in the availability of Fog devices (i.e., joining, leaving), as defined in Section III-D.

B. Fog Multilayer Partitioning

The *multilayer partitioning* of the Fog devices from a multilayer graph based on their network connections, processing, memory, and storage resources has three steps (see Section V-B and Algorithm 1):

- 1) *Layer partitioning* splits each layer $l \in L$ of the multilayer graph in a set $\mathcal{P}(l)$ of disjoint partitions that cluster the devices based on their resource types targeting a single objective such as transmission time, processing speed, and resource wastage (see Section III-B).
 - *Network layer l_0 partitioning* clusters the highly interconnected devices based on their network connections N .
 - *Processing layer l_1 partitioning* cluster Fog devices with a similar number of cores and processing speed.
 - *Memory l_2 , and storage l_3 layer partitioning* cluster Fog devices with similar memory and storage sizes.

- 2) *Graph compression* shrinks the disjoint partitions from the processing, memory, and storage layer partitions in a simpler representation associated with similar resources. The compressed graph enables the detection of overlapping partitions without analyzing individual devices resulting in considerable reductions in both time and cost.
- 3) *Feature partitioning* splits the compressed graph into disjoint partitions, where each partition clusters devices with similar average processing, memory, and storage sizes to address multiple objectives, such as minimizing transmission time, processing speed, and resource wastage.
- 4) *Incremental multilayer partitioning* updates the multilayer partitions, compressed graph, and feature partitions upon the temporal availability of the Fog devices and their resources (see Section V-C and Algorithm 2) to adaptively detect clusters in dynamic Fog with minimum cost and time.

C. M-RAP Placement

The placement phase maps all application sets AS_t arriving during the time interval Δt in two phases.

- 1) *Dynamic application placement* allocates resources to all services simultaneously to ensure their placement in the same network partition and satisfy their deadlines.
 - a) *Similarity placement* maps every service s_i onto the proper feature partition FP_k^t based on its deadline and resource requirements according to the placement function (see Section VI-A and Algorithm 3). This phase narrows the search space by exploring the partitions rather than all Fog devices.
 - b) *Monitoring and replacement* softens the hard deadline in critical cases of invalid placements and leaving devices and relocates the affected services to new feature partitions with minimal deadline violation.
- 2) *Service placement* assigns a Fog device $d_j^t = \mu_t(s_i)$ in the selected feature partition to each service s_i while ensuring the placement of the entire application within the same network layer partition (see Section VI-B and Algorithm 4).

V. FOG MULTILAYER PARTITIONING

This section describes the multilayer partitioning method.

A. Modularity

We convert multilayer graph of Fog resources into multilayer partitions using *modularity* [30] metric $Q \in [-1, 1]$ to measure their connectivity strength:

$$Q = \frac{1}{2W} \cdot \sum_{l \in L} \sum_{v \in L} \sum_{d_j \in l} \sum_{d_q \in l} \left\{ \left(w_{jq}^{(l)} - \frac{\sigma_j^l \cdot \sigma_q^l}{2W_l} \right) \cdot \alpha_{l'} + B_{jq} \cdot E_{l'}^q \right\} \cdot \lambda_{jq},$$

- $\sigma_j^{(l)} = \sum_{d_q \in l} w_{jq}^{(l)}$ and $\sigma_q^{(l)} = \sum_{d_j \in l} w_{jq}^{(l)}$ are the connectivity strengths of d_j and d_q with the layer l 's devices;
- $W_l = \sum_{d_j \in l} \sum_{d_q \in l} w_{jq}^{(l)}$ is the total sum of the intra-layer edges' weights between devices d_j and d_q in layer $l \in L$;
- $W = \sum_{l \in L} W_l$ is the total sum of the intra-layer edges' weights between Fog devices, $\forall l \in L$;
- $E_{ll'}^j$ indicates the number of inter-layer edges of the Fog device d_j from layer l to layer l' ;
- $\alpha_{ll'}$ is equal to 1 if $l = l'$ and 0 otherwise;
- B_{jq} is equal to 1 if $j = q$ and 0 otherwise;
- λ_{jq} is equal to 1 if devices d_j and d_q belong to the same partition, otherwise 0.

$Q \leq 0$ represents low-quality partitions of disassortative Fog devices with sparse connections among them.

$Q > 0$ represents high-quality partitions with better connectivity strength among densely connected Fog devices. Hence, the goal is to find a set of partitions in a multilayer graph with the highest modularity ($Q \rightarrow 1$).

B. Multilayer Partitioning

We convert the multilayer graph of Fog resources into a *multilayer partition* $(\mathcal{P}(l), \mathcal{P}(G_P))$ of layer $\mathcal{P}(l)$ and feature partitions $\mathcal{P}(G_P)$ in three steps.

- 1) *Layer partitioning* defines a set of disjoint partitions $\mathcal{P}(l) = \{p_1, p_2, \dots\}$ in a layer l , and employs the Louvain clustering [31] and the modularity [30] metric to obtain high-quality partitions with densely connected devices. The Louvain algorithm applies two phases in multiple iterations until achieving a partition with the maximum modularity.
 - a) We consider each Fog device d_j in a layer l as a single partition, calculate the modularity and define it as Q_{max} . Afterwards, we consider all its neighboring devices d_q (i.e., $(d_j, d_q) \in E_{ll}$ and $(d_q, d_j) \in E_{ll}$) and calculate the modularity Q_1 by considering the new possible partition. If the gain in modularity is positive (i.e., $Q_1 - Q_{max} > 0$), we place d_j and d_q in the same partition and consider Q_1 as Q_{max} . We repeat this step sequentially for all devices in the layer l until no further modularity gain is possible.
 - b) We consider all partitions from the first phase as nodes of each layer l , connected according to the edges between their devices. The edges between the Fog devices in the same partition represent self-loops. This new graph and its maximum modularity represent the input to the next iterative step starting with the first phase.
- 2) *Graph compression* merges similar resources from a partition in a single node with an average speed or size in the processing, memory, and storage layers. This high-level intermediate representation of the partitions associated with similar resources automates the detection of overlapping partitions without a detailed analysis of individual devices. A *compressed graph* $G_P = (V_P, E_P)$ corresponding to a multilayer graph $G = (D, \mathcal{E}, L)$ consists of two sets:
 - a) *Layer partition nodes* are the union in processing (l_1), memory (l_2), and storage (l_3) layers: $V_P = \bigcup_{l \in L \wedge l > 0} \mathcal{P}(l)$.
 - b) *Inter-layer partition edges* represent connections between a partition p in a layer l and a partition p' in a layer $l' \neq l$, such that there is at least one inter-layer edge $(d_q, d_q) \in E_{ll'}$ in the original graph G between a device $d_q \in p \in \mathcal{P}(l)$ and a device $d_q \in p' \in \mathcal{P}(l')$:

$$E_P = \{(p, p') \in \mathcal{P}(l) \times \mathcal{P}(l') \mid \forall l \neq l' \in L$$

$$\wedge \exists (d_q, d_q) \in E_{ll'} \wedge d_q \in p \wedge d_q \in p'\}.$$
- 3) *Feature partitioning* splits a compressed graph $G_P = (V_P, E_P)$ in a set $\mathcal{P}(G_P)$ of disjoint feature partitions (exhibiting similar features) by applying several Louvain clustering steps to achieve maximum modularity (similar to the layer partitioning from Section I). A *feature* of a layer partition $p \in V_P$ is a quadruple $F_p = (R_{p1}, R_{p2}, R_{p3}, R_{p4})$ with the average number of cores, memory and storage sizes, and processing speed across all devices in p .

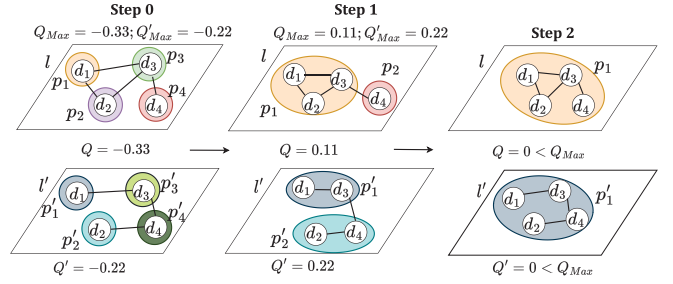


Fig. 4. Modularity (Q) computation at steps 1 and 2 for the multilayer graph from Fig. 2.

- a) *Layer partition nodes* are the union in processing (l_1), memory (l_2), and storage (l_3) layers: $V_P = \bigcup_{l \in L \wedge l > 0} \mathcal{P}(l)$.
 - b) *Inter-layer partition edges* represent connections between a partition p in a layer l and a partition p' in a layer $l' \neq l$, such that there is at least one inter-layer edge $(d_q, d_q) \in E_{ll'}$ in the original graph G between a device $d_q \in p \in \mathcal{P}(l)$ and a device $d_q \in p' \in \mathcal{P}(l')$:

$$E_P = \{(p, p') \in \mathcal{P}(l) \times \mathcal{P}(l') \mid \forall l \neq l' \in L$$

$$\wedge \exists (d_q, d_q) \in E_{ll'} \wedge d_q \in p \wedge d_q \in p'\}.$$
- 3) *Feature partitioning* splits a compressed graph $G_P = (V_P, E_P)$ in a set $\mathcal{P}(G_P)$ of disjoint feature partitions (exhibiting similar features) by applying several Louvain clustering steps to achieve maximum modularity (similar to the layer partitioning from Section I). A *feature* of a layer partition $p \in V_P$ is a quadruple $F_p = (R_{p1}, R_{p2}, R_{p3}, R_{p4})$ with the average number of cores, memory and storage sizes, and processing speed across all devices in p .
- Example 4.* We partition the multilayer graph in Fig. 2.
- 1) *Layer partitioning*: The Louvain algorithm finds the highest modularity partitions in each layer in two steps.
 - a) We create four partitions $\mathcal{P}(l) = \{p_1, p_2, p_3, p_4\}$ in the layer l and other four $\mathcal{P}(l') = \{p'_1, p'_2, p'_3, p'_4\}$ in the layer l' (see Fig. 4). Each partition in $\mathcal{P}(l)$ and $\mathcal{P}(l')$ consists of one Fog device (i.e., d_1, d_2, d_3, d_4) with the modularities $Q_{max} = -0.33$ and $Q'_{max} = -0.27$. Afterward, we consider all the neighboring devices and create two partition sets $\mathcal{P}(l) = \{p_1, p_2\}$ and $\mathcal{P}(l') = \{p'_1, p'_2\}$ with a positive modularity gain (i.e., $Q_1 = 0.11 > Q_{max}$, $Q'_1 = 0.22 > Q'_{max}$), indicating better device connectivity in each layer partition. Thus, we select the partitions $\mathcal{P}(l)$ and $\mathcal{P}(l')$, update the maximum modularities (i.e., $Q_{max} = Q_1$, $Q'_{max} = Q'_1$), and consider p_1, p_2, p'_1, p'_2 as nodes in a new multilayer graph.
 - b) We build one partition in each layer considering the partitions p_1, p_2, p'_1, p'_2 with the highest modularities Q_{max} and Q'_{max} from the first step, and the neighboring nodes, $\mathcal{P}(l) = \{p_1\}$ and $\mathcal{P}(l') = \{p'_1\}$, with the modularities $Q_2 = Q'_2 = 0$ (see Fig. 4). As the maximum modularities from the first step are positive for

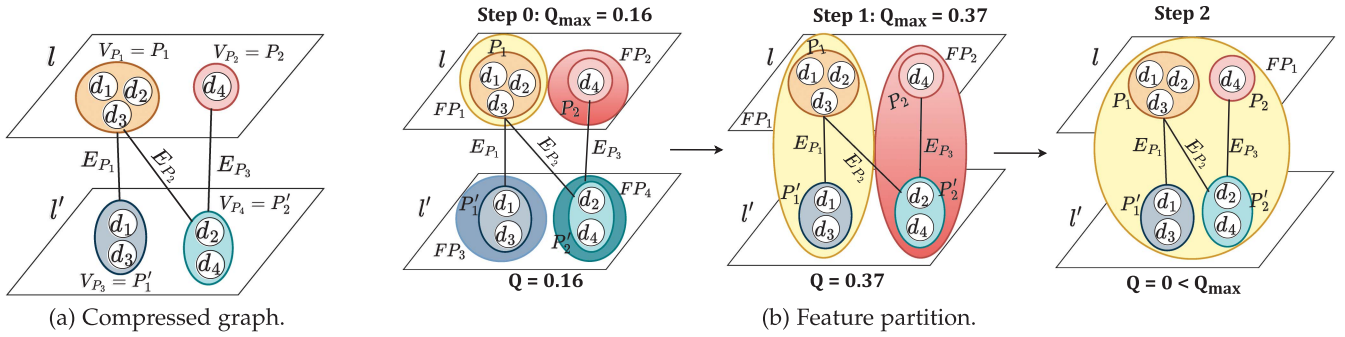


Fig. 5. Feature partitioning example.

both layers, we consider them as the highly connected partition output.

- 2) *Graph compression*: Fig. 5(a) illustrates a compressed graph representing the four partitions in the l and l' layers from Fig. 4 as nodes: $V_P = \{p_1, p_2, p'_1, p'_2\}$. Similar to the nodes, we compress the edges between two partitions: $E_p = \{(p_1, p'_1), (p_1, p'_2), (p_2, p'_2)\}$.
- 3) *Feature partitioning*: Fig. 5(b) iterates two feature partitioning steps on the compressed graph G_P from Fig. 5(a).
 - a) We create a set of four feature partitions $\mathcal{P}(G_P) = \{FP_1, FP_2, FP_3, FP_4\}$ with the modularity $Q_{max} = 0.16$, where each feature partition FP_k consists of a single layer partition (i.e., p_1, p_2, p'_3, p'_4). Afterward, we consider the neighboring partitions to generate a new feature partition set $\mathcal{P}(G_P) = \{FP_1, FP_2\}$ with a positive modularity gain $Q = 0.37 > Q_{max}$, which becomes the maximum modularity $Q_{max} = Q$.
 - b) We start from the feature partition set $\mathcal{P}(G_P)$ with the maximum modularity from the first step. We iteratively check the neighboring partitions of each feature partition $FP_k \in \mathcal{P}(G_P)$ considered as a node of the new graph and obtain a single partition FP_1 with lower modularity $Q = 0$. We select the feature partition set $\mathcal{P}(G_P) = \{FP_1, FP_2\}$ with the highest modularity from the first step as output.

C. Incremental Multilayer Partitioning

We update the multilayer partitions $(\mathcal{P}_t(l), \mathcal{P}_t(G_P))$ during the time intervals Δt in three steps to maximize the modularity in response to the availability of Fog devices.

- 1) *Incremental layer partitioning* updates the previous layer partitions $\mathcal{P}_{t-1}(l)$ to $\mathcal{P}_t(l)$ based on the *layer partition changes* $\Delta\mathcal{P}_t(l)$ computed from the multilayer graph changes $\Delta G_t = (GJ_t, GL_t)$ during $\Delta t = [t-1, t]$.
 - a) *Incremental growth* $GJ_t = (DJ_t, EJ_t)$ upon every joining device $d_{n+1} \in DJ_t$ has three cases.
 - Case-1: If d_{n+1} has no intra-layer edges in the layer $l \in L$, we create an isolated partition $p = d_{n+1} \in \Delta\mathcal{P}_t(l)$ in l , and consider other layer partitions unchanged.
 - Case-2: If d_{n+1} has intra-layer edges in the same layer partition $p \in \mathcal{P}_t(l)$, we add d_{n+1} to the partition

$p \in \Delta\mathcal{P}_t(l)$ and consider the other layer partitions unchanged.

Case-3: If d_{n+1} has intra-layer edges in more layer partitions, we add it to the partition $p \in \Delta\mathcal{P}_t(l)$ with the highest aggregated similarity score $\Delta w_{(n+1)p}^l = \sum_{d_j \in p} w_{(n+1)j}^l$ across all its edges, where m is the number of devices in p .

- b) *Incremental shrink* $GL_t = (DL_t, EL_t)$ upon every leaving device $d_{n-1} \in DL_t$ has two cases.

Case-1: If $d_{n-1} \in p \in \mathcal{P}_t(l)$ has no edges, we delete it from its associated partition $p \in \Delta\mathcal{P}_t(l)$ and consider the other layer partitions unchanged.

Case-2: If d_{n-1} has several connecting edges, we restructure all the devices in the partition $p \in \Delta\mathcal{P}_t(l)$ containing d_{n-1} and its neighboring partitions $p' \in \Delta\mathcal{P}_t(l)$ into single partitions, and apply the Louvain algorithm.

- 2) *Incremental graph compression* updates the compressed graph $G_p^t = (V_p^t, E_p^t)$ at the time interval Δt based on the layer partition changes $\Delta\mathcal{P}_t(l)$ in the processing, memory, and storage layers during the time interval Δt . We compute the *compressed graph changes* $\Delta G_p^t = (\Delta V_p^t, \Delta E_p^t)$ by merging the layer partition changes $\Delta\mathcal{P}_t(l)$ and their inter-layer edges into two sets of single nodes ΔV_p^t and edges ΔE_p^t .
- 3) *Incremental feature partitioning* updates the feature partitions $\mathcal{P}_t(G_p)$ at the time interval Δt using the compressed graph changes $\Delta G_p^t = (\Delta V_p^t, \Delta E_p^t)$ and the feature partitions during the interval $\Delta t = [t-1, t]$. We restructure the layer partition changes ΔV_p^t in the processing, memory, and storage layers to single partitions and repeat the feature partitioning steps for the compressed graph changes ΔG_p^t (see Sections II and III). We leave the other partitions unchanged.

Example 5. We update partitions upon incremental changes in the multilayer graph $\Delta G_t = (GJ_t, GL_t)$ from Section III-D.

- 1) *Incremental layer partitioning* updates the partitions based on the incremental growth and shrink in two steps:

GJ_t : The joining device d_5 in the layer l belongs to the Case-3, since it has two edges to the layer partitions $p_1, p_2 \in \mathcal{P}(l)$ with the aggregated similarity scores of $w_{51}^l = 0.5$ and $w_{52}^l = 1$. We, therefore, assign d_5 to the

layer partition p_2 with the highest score and update the layer partition changes $\Delta\mathcal{P}_t(l) = p_2$. The joining device d_5 in the layer l' has two edges to the same layer partition $p'_2 \in \mathcal{P}(l')$ and belongs to the Case-2. Thus, we add it to the layer partition p'_2 and update the layer partition changes $\Delta\mathcal{P}_t(l') = p'_2$.

GL_t : The leaving device d_1 in the layers l and l' represents the Case-2 in Section V-C. We restructure the layer partitions $p_1 \in \mathcal{P}(l)$ and $p'_1 \in \mathcal{P}(l')$ to single partitions and update them to $p_1 = \{d_2, d_3\}$ and $p'_1 = \{d_2\}$ using the Louvain algorithm. Finally, we update the layer partition changes to $\Delta\mathcal{P}_t(l) = \{p_1, p_2\}$ and $\Delta\mathcal{P}_t(l') = \{p'_1, p'_2\}$.

- 2) *Incremental graph compression* joins the devices in the layer partitions $\{p_1, p_2\} \in \Delta\mathcal{P}_t(l)$ and $\{p'_1, p'_2\} \in \Delta\mathcal{P}_t(l')$ into the set of single nodes $\Delta V_p^t = \{p_1, p_2, p'_1, p'_2\}$. Similarly, it compresses the inter-layer edges between these partitions to single edges: $\Delta E_P^t = \{E_{P_1}, E_{P_2}, E_{P_3}\}$.
- 3) *Incremental feature partitioning* applied on the compress graph changes $\Delta G_P^t = (\Delta V_p^t, \Delta E_P^t)$ generates: $FP_1 = \{p_1, p'_1\}$ and $FP_2 = \{p_2, p'_2\}$.

D. Multilayer Resource Partitioning Algorithm

Algorithm 1 receives the following input parameters:

- 1) a Fog multilayer graph with four layers L ,
- 2) a set of Fog devices D and their processing, memory, and storage resources, and
- 3) the inter-layer and intra-layer edges \mathcal{E} .

First, line 1 initializes five empty sets corresponding to the partitions in the network (l_0), processing (l_1), memory (l_2) and storage (l_3) layers, and the feature partitions. Line 2 clusters densely connected Fog devices in the same network layer partition $\mathcal{P}(l_0)$. Similarly, lines 3–5 partition the Fog devices in the processing $\mathcal{P}(l_1)$, memory $\mathcal{P}(l_2)$, and storage $\mathcal{P}(l_3)$ layers (see Section I). Line 6 creates a compressed graph $G_P(V_P, E_P)$ using the inter-layer edges between the processing, memory, and storage layers, where $V_P = \{\mathcal{P}(l_1), \mathcal{P}(l_2), \mathcal{P}(l_3)\}$ (see Section II). Afterward, lines 7 computes the feature F_P of each partition $p \in V_P$ as their devices' $d_j \in p$ average number of cores, memory, storage sizes, and processing speed. Line 8 performs feature partitioning on the compressed graph, as presented in Section III. Finally, line 9 returns the feature partition set $\mathcal{P}(G_P)$ and the set of partitions in the network layer $\mathcal{P}(l_0)$.

layerPartition function clusters devices in disjoint partitions $\mathcal{P}(l)$ in a layer l (line 11), initialized with the empty set in line 12. Afterward, lines 13–17 assign each Fog device d_j in layer l to a single partition and add it to the layer partitions $\mathcal{P}(l)$. Lines 19–26 iterate all partitions p_j and their neighbor partitions p_q and calculate the modularity considering the new possible partitions $p_j \cup p_q$ (line 25). Afterward, line 21 iterates as long as the modularity gain Q is greater than modularity Q_{max} , merges the partition p_j with partition p_q in line 22 and considers Q as Q_{max} in line 23. Otherwise, it stops merging the partition p_j with its neighbor p_q . Finally, line 27 returns layer partitions $\mathcal{P}(l)$.

Algorithm 1. Multilayer Resource Partitioning.

```

Input :  $G = (D, \mathcal{E}, L)$ : Fog multilayer graph
          $L = \{l_0, l_1, l_2, l_3\}$ : Fog layers
          $D = \{d_j | d_j = (R_{j1}, R_{j2}, R_{j3}, R_{j4})\}$ : Fog devices
          $\mathcal{E} = \{E_{ll'} | \forall l, l' \in [0, L]\}$ : inter- and intra-layer edges
Output:  $\mathcal{P}(G_P)$ : feature partition set
          $\mathcal{P}(l_0)$ : network layer partition set
1   $\mathcal{P}(l_0) \leftarrow \emptyset; \mathcal{P}(l_1) \leftarrow \emptyset; \mathcal{P}(l_2) \leftarrow \emptyset; \mathcal{P}(l_3) \leftarrow \emptyset; \mathcal{P}(G_P) \leftarrow \emptyset$ 
2   $\mathcal{P}(l_0) \leftarrow \text{layerPartition}(D, E_{l_0l_0}, l_0)$ 
3   $\mathcal{P}(l_1) \leftarrow \text{layerPartition}(D, E_{l_1l_1}, l_1)$ 
4   $\mathcal{P}(l_2) \leftarrow \text{layerPartition}(D, E_{l_2l_2}, l_2)$ 
5   $\mathcal{P}(l_3) \leftarrow \text{layerPartition}(D, E_{l_3l_3}, l_3)$ 
6   $G(V_P, E_P) \leftarrow$ 
    $\text{compressGraph}(\mathcal{P}(l_1), \mathcal{P}(l_2), \mathcal{P}(l_3), E_{l_1l_2}, E_{l_1l_3}, E_{l_2l_3})$ 
7   $fList \leftarrow \text{getFeatures}(V_P)$ ;
8   $\mathcal{P}(G_P) \leftarrow \text{featurePartition}(G_P(V_P, E_P), fList)$ 
9  return  $(\mathcal{P}(G_P), \mathcal{P}(l))$ 
10
11 Function  $\text{layerPartition}(D, E_{ll}, l)$ :
12    $\mathcal{P}(l) \leftarrow \emptyset$ 
13   forall  $d_j \in D$  do
14      $p_j \leftarrow \emptyset$ 
15      $p_j \leftarrow p_j \cup \{d_j\}$ 
16      $\mathcal{P}(l) \leftarrow \mathcal{P}(l) \cup p_j$ 
17   end
18    $Q_{max} \leftarrow \text{modularity}(\mathcal{P}(l), E_{ll}, l)$ 
19   forall  $p_j, p_q \in \mathcal{P}(l) \wedge (d_j \in p_j \wedge d_q \in p_q \wedge (d_q, d_j) \in E_{ll})$  do
20      $Q \leftarrow \text{modularity}(p_j \cup p_q, E_{ll}, l)$ 
21     while  $Q_{max} < Q$  do
22        $p_j \leftarrow p_j \cup p_q$ 
23        $Q_{max} \leftarrow Q$ 
24      $Q \leftarrow \text{modularity}(p_j \cup p_q, E_{ll}, l)$ 
25     end
26   end
27   return  $\mathcal{P}(l)$ 
28
29 Function  $\text{getFeatures}(V_P)$ :
30    $fList \leftarrow \emptyset$ ;
31   forall  $p \in V_P$  do
32      $R_{p1} \leftarrow \text{avg}_{d_j \in p} \{R_{j1}\}; R_{p2} \leftarrow \text{avg}_{d_j \in p} \{R_{j2}\};$ 
33      $R_{p3} \leftarrow \text{avg}_{d_j \in p} \{R_{j3}\}; R_{p4} \leftarrow \text{avg}_{d_j \in p} \{R_{j4}\}$ 
34      $fList \leftarrow fList \cup (R_{p1}, R_{p2}, R_{p3}, R_{p4})$ 
35   end
36   return  $fList$ 

```

getFeatures function computes the feature list of all partitions $p \in V_P$ (line 29), initialized with an empty list in line 30. Lines 31–33 iterate through partitions $p \in V_P$, calculate their feature as a quadruple of the average number of cores, memory and storage sizes, and processing speed across all their devices $d_j \in p$ (line 32) and adds it to feature list of all partitions $fList$ (line 33), returned in line 35.

Computational complexity of Algorithm 1 is $\mathcal{O}(|L| \cdot |E_{ll}| + |E_P|)$, where $|L|$ is the number of layers in Fog multilayer graph, $|E_{ll}|$ is the number of intra-layer edges in layer l , $|E_P|$ is the number of edges in compressed graph G_P , and $|L| \ll |E_P| \ll |E_{ll}|$. Hence, the algorithm has a linear complexity of $\mathcal{O}(|E_{ll}|)$.

E. Incremental Multilayer Partitioning Algorithm

Algorithm 2 has the following parameters:

- 1) a Fog multilayer graph during the time interval Δt with four layers L ,
- 2) the inter-layer and intra-layer edges \mathcal{E}_t ,
- 3) the incremental multilayer graph changes ΔG_t during the time interval $[t-1, t]$, and
- 4) the layer $\mathcal{P}_{t-1}(l)$ and feature partitions $\mathcal{P}_{t-1}(G_P)$ of the previous time interval.

First, line 1 initializes five empty sets corresponding to the partitions in the network (l_0), processing (l_1), memory (l_2), and

storage (l_3) layers, and the feature partitions $\mathcal{P}_t(G_p)$ during the time interval Δt . Lines 2–4 update the layer partitions and calculate the changes $\Delta\mathcal{P}_t$ in the network, processing, memory, and storage layers based on the incremental multilayer graph changes ΔG_t during the interval Δt and the previous layer partition $\mathcal{P}_{t-1}(l_i)$ (see Section V-C). Line 5 generates a compressed graph of partition changes in the processing, memory, and storage layers and their associated inter-layer edges. Line 6 calculates the feature F_p of layer partitions changes as the average number of cores, memory and storage size, and processing speed of their devices $d_j^t \in \Delta V_p^t$. Line 7 updates the feature partitions using the compressed graph changes $\Delta G_p^t(\Delta V_p^t, \Delta E_p^t)$, and the previous feature partitions $\mathcal{P}_{t-1}(G_p)$ (see Section V-C). Finally, line 8 returns the network layer and feature partitions ($\mathcal{P}_t(l_0), \mathcal{P}_t(G_p)$) during the time interval Δt .

dLayerPartition function updates the partitions upon changes in the Fog (line 10). Line 11 initializes the layer partition changes $\Delta\mathcal{P}_t$ and layer partitions $\mathcal{P}_t(l_i)$ with the empty set. Afterward, lines 12–14 iterate through all joining devices $d_j^t \in DJ_t$, and line 13 updates the partitions based on three incremental growth case conditions defined in Section V-C. Similarly, lines 15–17 iterate through all the leaving devices $d_j^t \in DL_t$, and line 16 updates the partitions based on the two incremental shrink case conditions explained in section V-C. Line 18 creates a new set of partitions $\mathcal{P}_t(l_i)$ based on layer partition changes $\Delta\mathcal{P}_t(l_i)$ and previous layer partitions $\mathcal{P}_{t-1}(l_i)$. Line 19 returns the layer partition changes $\Delta\mathcal{P}_t(l_i)$ and the updated layer partition $\mathcal{P}_t(l_i)$.

compressGraph function initializes the compressed node changes ΔV_p^t and their associated edges ΔE_p^t with the empty set in line 22. Lines 23–26 iterate through the partitions in the layer partition changes and add them together with their associated inter-layer edges to the compressed graph changes $\Delta G_p^t(\Delta V_p^t, \Delta E_p^t)$ returned in line 27.

dFeatPart function updates feature partitions $\mathcal{P}_t(G_p)$ upon incremental changes $\Delta\mathcal{P}_t(G_p)$ (line 29), initialized in line 30 with the empty set. Lines 31–33 iterate through changed partitions V_p^t in the compressed graph ΔG_p^t . Line 32 updates the previous feature partitions $\mathcal{P}_{t-1}(G_p)$ by repeating the feature partitioning step for every change V_p^t in the compressed graph, added to the feature partition changes $\Delta\mathcal{P}_t(G_p)$. Line 34 updates the feature partitions $\mathcal{P}_t(G_p)$ by adding the changes $\Delta\mathcal{P}_t(G_p)$ to the previous feature partitions $\mathcal{P}_{t-1}(G_p)$, returned in line 35.

Computational complexity of Algorithm 2 is $\mathcal{O}(|L| \cdot |EJ_{ll} + EL_{ll}| + |\Delta E_P|)$, where $|EJ_{ll}|$ and $|EL_{ll}|$ are the numbers of intra-layer edges of joining and leaving devices in the layer l , $|\Delta E_P|$ is the number of edges in the compressed graph changes ΔG_p , and $|L| \ll |\Delta E_P| \ll |EJ_{ll} + EL_{ll}|$. Hence, the algorithm has a linear complexity of $\mathcal{O}(|EJ_{ll} + EL_{ll}|)$, which is negligible compared to Algorithm 1 ($|EJ_{ll} + EL_{ll}| \ll |E_{ll}|$), since it only updates the partitions based on incremental changes.

VI. M-RAP PLACEMENT

This section describes the placement of application services in dynamically selected feature partitions based on two dynamic application and service placement algorithms.

Algorithm 2. Incremental Multilayer Partitioning.

Input : $L = \{l_0, l_1, l_2, l_3\}$: Fog layers
 $\mathcal{E}_t = \{E_{ll'}^t \mid \forall l, l' \in [0, L]\}$: inter- and intra-layer edges
 $\Delta G_t = (GJ_t, GL_t)$: multilayer graph changes
 $\mathcal{P}_{t-1}(G_P)$: previous feature partition set (Algorithm 1)
 $\mathcal{P}_{t-1}(l)$: previous layer partitions (Algorithm 1)

Output: $\mathcal{P}_t(G_P)$: new feature partition set
 $\mathcal{P}_t(l_0)$: new network layer partition set

```

1  $\mathcal{P}_t(l_0) \leftarrow \emptyset; \mathcal{P}_t(l_1) \leftarrow \emptyset; \mathcal{P}_t(l_2) \leftarrow \emptyset; \mathcal{P}_t(l_3) \leftarrow \emptyset; \mathcal{P}_t(G_P) \leftarrow \emptyset$ 
2 forall  $l_i \in L$  do
3    $(\mathcal{P}_t(l_i), \Delta\mathcal{P}_t(l_i)) \leftarrow \text{dLayerPartition}(\Delta G_t, \mathcal{P}_{t-1}(l_i), l_i)$ 
4 end
5  $\Delta G_p^t(\Delta V_p^t, \Delta E_p^t) \leftarrow$ 
    $\text{compressGraph}(\Delta\mathcal{P}_t(l_1), \Delta\mathcal{P}_t(l_2), \Delta\mathcal{P}_t(l_3), E_{l_1l_2}^t, E_{l_1l_3}^t, E_{l_2l_3}^t)$ 
6  $fList \leftarrow \text{fValue}(\Delta V_p^t)$ 
7  $\mathcal{P}_t(G_P) \leftarrow \text{dFeatPart}(\Delta G_p^t(\Delta V_p^t, \Delta E_p^t), \mathcal{P}_{t-1}(G_P), fList)$ 
8 return  $(\mathcal{P}_t(l_0), \mathcal{P}_t(G_P))$ 
9
10 Function  $\text{dLayerPartition}(\Delta G_t, \mathcal{P}_{t-1}(l_i), l_i)$ :
11    $\Delta\mathcal{P}_t(l_i) \leftarrow \emptyset; \mathcal{P}_t(l_i) \leftarrow \emptyset$ 
12   forall  $d_j^t \in DJ_t \in GJ_t$  do
13      $\Delta\mathcal{P}_t(l_i) \leftarrow \Delta\mathcal{P}_t(l_i) \cup \text{GrowthPar}(GJ_t, \mathcal{P}_{t-1}(l_i), l_i)$ 
14   end
15   forall  $d_j^t \in DL_t \in GL_t$  do
16      $\Delta\mathcal{P}_t(l_i) \leftarrow \Delta\mathcal{P}_t(l_i) \cup \text{ShrinkPar}(GL_t, \mathcal{P}_{t-1}(l_i), l_i)$ 
17   end
18    $\mathcal{P}_t(l_i) \leftarrow \Delta\mathcal{P}_t(l_i) \cup \mathcal{P}_{t-1}(l_i)$ 
19   return  $(\Delta\mathcal{P}_t(l_i), \mathcal{P}_t(l_i))$ 
20
21 Function
    $\text{compressGraph}(\Delta\mathcal{P}_t(l_1), \Delta\mathcal{P}_t(l_2), \Delta\mathcal{P}_t(l_3), E_{l_1l_2}^t, E_{l_1l_3}^t, E_{l_2l_3}^t)$ :
22    $\Delta V_p^t \leftarrow \emptyset; \Delta E_p^t \leftarrow \emptyset$ 
23   forall  $\mathcal{P}_t(l) \in \Delta\mathcal{P}_t(l) \wedge E_p^t \in (p, p') \in \mathcal{P}_t(l) \times \mathcal{P}_t(l')$  do
24      $\Delta V_p^t \leftarrow \Delta V_p^t \cup \mathcal{P}_t(l)$ ;
25      $\Delta E_p^t \leftarrow \Delta E_p^t \cup E_p^t$ ;
26   end
27   return  $\Delta G_p^t(\Delta V_p^t, \Delta E_p^t)$ 
28
29 Function  $\text{dFeatPart}(\Delta G_p^t(\Delta V_p^t, \Delta E_p^t), \mathcal{P}_{t-1}(G_P), fList)$ :
30    $\mathcal{P}_t(G_P) \leftarrow \emptyset; \Delta\mathcal{P}_t(G_P) \leftarrow \emptyset$ 
31   forall  $V_p^t \in \Delta V_p^t \in \Delta G_p^t$  do
32      $\Delta\mathcal{P}_t(G_P) \leftarrow \Delta\mathcal{P}_t(G_P) \cup$ 
        $\text{IncPar}(V_p^t, \mathcal{P}_{t-1}(G_P), fList)$ 
33   end
34    $\mathcal{P}_t(G_P) \leftarrow \Delta\mathcal{P}_t(G_P) \cup \mathcal{P}_{t-1}(G_P)$ ;
35   return  $\mathcal{P}_t(G_P)$ 

```

A. Dynamic Application Placement Algorithm

The dynamic application placement algorithm maps each application service during a time interval to a feature partition with the highest fitness. We define the *fitness* of a feature partition FP_k^t for a service $s_i \in \mathcal{S}$ based on the similarity of its Fog devices to the service resource demands and the lowest transmission time to the requesting user u :

$$\begin{aligned}
 \text{Fit}(FP_k^t, s_i, u, \mathcal{T}) &= \max_{p \in FP_k^t} (\text{Sim}(p, s_i) + \text{Rank}(p, TR_{uj})),
 \end{aligned}$$

- 1) $\max_{p \in FP_k^t} \{\text{Sim}(p, s_i)\}$ is the maximum partition similarity between the service demand $s_i = (r_{i1}, r_{i2}, r_{i3}, r_{i4})$ with the feature $F_p = (R_{p1}^t, R_{p2}^t, R_{p3}^t, R_{p4}^t)$ in FP_k^t at the time interval Δt , calculated using the euclidean distance in the $[0, 1]$ interval. Placing a service to a device in a partition with the highest similarity satisfies the service demands and avoids resource wastage on devices with higher capacity.
- 2) $\text{Rank}(p, TR_{uj})$ ranks the partition p based on the transmission time TR_{uj} between the device $d_j^t \in p \in FP_k^t$ and the user u requesting the service s_i . The partition with the

highest rank, normalized in the $[0, 1]$ interval, contains the device d_j^t with the lowest transmission time to the user.

Algorithm 3 receives five parameters for computing feature partitions that satisfy the application deadline Θ and individual service resource demands:

- 1) an application workload W over t time intervals,
- 2) the feature partition set $\mathcal{P}_t(G_P)$,
- 3) the network partition set $P_t(l_0)$,
- 4) the message transmission times \mathcal{T} between the users and the Fog devices, and
- 5) the leaving devices during the time interval Δt .

First, sorts the applications AS_t requested in time interval Δt to prioritize those with the lowest deadline. The algorithm places the applications in two steps.

- 1) *Similarity placement* (lines 2–4) selects the appropriate feature partitions for all applications $A \in AS_t$ requested during the time interval Δt and place their services $s_i \in S$ on the Fog devices that satisfy their resource demands close to the requesting users. In this first stage, the `placeApp` function considers the service deadlines and declares invalid placement $\mu_t(s_i) = \emptyset$ in case of any violation.
- 2) *Monitoring* (lines 5–13) iterates through all executing services and takes two actions:
 - a) *release* resources of completed tasks (lines 6–8);
 - b) *replace* mapping devices in case of leaving devices (lines 9–10) or invalid placements (lines 11–12), by invoking again the `placeApp` function to select a new feature partition for placing the affected service close to its requesting user satisfying the resource demands. In the case of invalid placement, the new invocation relaxes the hard deadline (set to infinity) and places the affected service on the fastest available device.

The algorithm returns an array of placement functions $\mu L[W]$ in line 13.

`placeApp` function selects the feature partition (line 16) that satisfies the resource demand of each service $s_i \in S$ (line 17). Lines 19–22 iterate through each feature partition $FP_k^t \in \mathcal{P}_t(G_P)$, calculate its fitness to s_i , and insert it in an `fpRank` list in descending fitness order (line 20). Line 21 sorts the devices $d_j^t \in FP_k^t$ in ascending order based on their transmission time TR_{uj} to user u and stores them in a matrix `dMatr`, where each row contains the transmission time to Fog devices in a feature partition $FP_k^t \in \mathcal{P}_t(G_P)$. Line 23 invokes a service placement function (see *Algorithm 4*) that maps all application services $s_i \in S$ onto devices with appropriate features in the same network layer partition. The function returns the device $\mu_t(s_i)$ in line 25, which updates the application placement μ_t returned in line 23.

Computational complexity of *Algorithm 3* is $\mathcal{O}(|S| \cdot |\mathcal{P}_t(G_P)|)$, where $|S|$ is the number of services and $|\mathcal{P}_t(G_P)|$ is the number of feature partitions at time interval Δt . The algorithm reduces the service placement complexity by clustering similar devices into a feature partition. Therefore, it only searches for appropriate feature partitions instead of iterating through all devices D , where $|\mathcal{P}_t(G_P)| \ll |D|$. Since

Algorithm 3. Dynamic Application Placement.

```

Input :  $W = \{AS_0, \dots, AS_t\}$ : dynamic application workload
          $\mathcal{P}_t(G_P)$ : feature partition set
          $\mathcal{P}_t(l_0)$ : network layer partition set
          $\mathcal{T} = \{TR_{uj}^t \mid (u, d_j^t) \in \mathcal{U} \times \mathcal{D}_t\}$ : transmission times
          $DL_t$ : leaving devices
Output:  $\mu L[W]$ : array of  $\mu L[A]$  service placements,  $\forall A \in W$ ;
1  $AS_t \leftarrow \text{sortApp}(AS_t, \Theta)$ 
2 forall  $A : (S, M, \Theta, u) \in AS_t$  do
3    $\mu L[A] \leftarrow \text{placeApp}(S, \Theta, u, \mathcal{T}, \mathcal{P}_t(G_P), P_t(l_0))$ 
4 end
5 forall  $A : (S, M, \Theta, u) \in W \wedge s_i \in S$  do
6   if  $\text{status}(s_i) = \text{completed}$  then
7      $S \leftarrow S \setminus s_i$ 
8      $(R_{j1}^t, R_{j2}^t, R_{j3}^t) \leftarrow (R_{j1}^t + r_{i1}, R_{j2}^t + r_{i2}, R_{j3}^t + r_{i3})$ 
9   else if  $\mu_t(s_i) \in DL_t$  then
10     $\mu L[A] \leftarrow \text{placeApp}(\{s_i\}, \{\theta_i\}, u, \mathcal{T}, \mathcal{P}_t(G_P), P_t(l_0))$ 
11  else if  $\mu_t(s_i) = \emptyset$  then
12     $\mu L[A] \leftarrow \text{placeApp}(\{s_i\}, \{\infty\}, u, \mathcal{T}, \mathcal{P}_t(G_P), P_t(l_0))$ 
13 end
14 return  $\mu L$ 
15
16 Function placeApp( $S, \Theta, u, \mathcal{T}, \mathcal{P}_t(G_P), P_t(l_0)$ ):
17   forall  $s_i \in S$  do
18      $fpRank \leftarrow \emptyset$ 
19     forall  $FP_k^t \in \mathcal{P}_t(G_P)$  do
20        $fpRank \leftarrow \text{insert}(fpRank, \text{Fit}(FP_k^t, s_i, u, \mathcal{T}))$ 
21        $dMatr[FP_k^t] \leftarrow \text{sortDev}(FP_k^t, \mathcal{T})$ 
22     end
23      $\mu_t(s_i) \leftarrow \text{placeService}(s_i, \theta_i, P_t(l_0), fpRank, dMatr)$ 
24   end
25 return  $\mu_t$ 

```

Algorithm 4. Service Placement.

```

Input:  $s_i = (r_{i1}, r_{i2}, r_{i3}, r_{i4})$ : service for placement
          $\theta_i$ : absolute service deadline
          $\mathcal{P}_t(l_0)$ : network layer partition set
          $fpRank$ : fitness-ranked feature partition set
          $dMatr[FP_k^t]$ : sorted Fog devices,  $\forall FP_k^t \in fpRank$ 
1 Function placeService( $s_i, \theta_i, P_t(l_0), fpRank, dMatr$ ):
2   if  $s_i = s_1$  then
3      $p_1 \leftarrow \text{getNetPartition}(\mu_t(s_1), P_t(l_0))$ 
4   forall  $FP_k^t \in fpRank$  do
5      $dList \leftarrow dMatr[FP_k^t]$ 
6     forall  $d_j^t = (R_{j1}^t, R_{j2}^t, R_{j3}^t, R_{j4}^t) \in dList$  do
7        $p_i \leftarrow \text{getNetPartition}(d_j^t, P_t(l_0))$ 
8       if  $p_1 = p_i \wedge RT_{ij}^t \leq \theta_i \wedge r_{i1} \leq R_{j1}^t \wedge r_{i2} \leq R_{j2}^t \wedge r_{i3} \leq R_{j3}^t$  then
9          $\mu_t(s_i) \leftarrow d_j^t$ 
10         $(R_{j1}^t, R_{j2}^t, R_{j3}^t) \leftarrow (R_{j1}^t - r_{i1}, R_{j2}^t - r_{i2}, R_{j3}^t - r_{i3})$ 
11      return  $\mu_t(s_i)$ 
12   end
13    $\mu_t(s_i) \leftarrow \emptyset$ 
14 return  $\mu_t(s_i)$ 

```

$|\mathcal{P}_t(G_P)| \ll |\mathcal{S}|$, the algorithm has a linear complexity of $\mathcal{O}(|S|)$.

B. Service Placement Algorithm

The service placement algorithm allocates a Fog device to each application service in the selected feature partitions. Placing the services of an application across tightly connected Fog devices in the same network layer partition during a time interval brings two advantages:

- 1) less network instability due to alternative connections between Fog devices;
- 2) lower transmission time between devices inside a partition, which reduces application response time.

TABLE IV
CARINTHIAN COMPUTING CONTINUUM (C^3) TESTBED

Layer		Cloud					Fog					
Provider		AWS	Google Cloud		AI / Exoscale			University of Klagenfurt (AAU)				
Instance type		t2.xlarge	n2.standard-4	large	medium	small	tiny	large	medium	N/N	RPI4	RPI3
Location		Virginia	Frankfurt	Sofia	Klagenfurt	Vienna	Munich, Zurich	Klagenfurt				
Device	Process. speed (MIPS)	11 200	10 100	14 000	7200	7100	3700	58 000	21 700	4080	5100	3500
	Processing (cores)	4	4	4	2	2	1	12	8	4	4	4
	Memory (GB)	16	16	8	4	2	1	32	16	4	4	1
	Storage (GB)	8	8	10	10	10	10	32	32	16	16	16
Net.	Bandwidth (Mbit/s)	100–710	500–800		450–850			300–920				
	Latency (ms)	15–100	10–30		7–28			1–2				

Algorithm 4 receives the following input parameters:

- 1) a service s_i to place on a Fog device $\mu_t(s_i)$ in the same network layer partition as the other application services;
- 2) an absolute deadline θ_i for a service s_i ,
- 3) a network layer partition set $P_t(l_0)$,
- 4) a feature partition set $fpRank$ ranked based on the fitness to s_i , and
- 5) a sorted list of devices based on their transmission time in each feature partition $FP_k^t \in fpRank$.

Lines 2–3 extract the network layer partition of the first service placement $\mu_t(s_1)$ in p_1 (if available). Lines 4–12 iterate through the feature partitions $FP_k^t \in fpRank$ at the time interval Δt in descending order of their fitness. Afterward, line 5 extracts the set of Fog devices $dList$ in each feature partition FP_k^t sorted by the transmission times to the requesting user. To place the service s_i onto a Fog device, lines 6–12 iterate through each device $d_j^t \in dList$ and line 7 extracts its network layer partition in p_i . If this partition is the same as p_1 and the device d_j^t meets the resource constraints of the service s_i (including its deadline θ_i), line 9 performs the placement and line 10 updates the available resources of device d_j^t . If no service placement on the same network partition is possible, line 14 assigns an invalid device. Lines 11 and 14 return the placement result.

Computational complexity of Algorithm 4 is $\mathcal{O}(1)$ in the best case, by placing a service onto a feature partition FP_k^t with the highest fitness and the device with the lowest user transmission time. If no device with sufficient resources exists, the algorithm searches for other feature partitions and their associated devices, leading to a worst-case complexity of $\mathcal{O}(|P_t(G_p)| \cdot |FP_K^t|)$, where $|P_t(G_p)|$ is the number of feature partitions during the time interval Δt and $|FP_K^t|$ is the number of devices in the feature partition FP_K^t . Since $|P_t(G_p)| \ll |FP_K^t|$, the algorithm has a linear worst case complexity of $\mathcal{O}(|FP_K^t|)$, where $|FP_K^t| \ll |D|$.

VII. EXPERIMENTAL SETUP

We perform experiments based on a real computing testbed and real-world applications and compare the results against three related methods. We first validate M-RAP using simulation in Section VIII followed by a real testbed in Section IX.

A. Carinthian Computing Continuum

We used the Carinthian Computing Continuum (C^3) testbed consisting of eight heterogeneous virtual resource instances from four providers distributed in seven geographical locations across the Cloud and Fog layers, displayed in Table IV.

Cloud resources consist of virtualized large instances provisioned on-demand on the Exoscale data centers (<https://www.exoscale.com/compute/>) in Sofia.

Fog devices comprise Exoscale provider instances in the data centers of the A1 network operator in Vienna, Zurich, and Munich with a 10Gbit/s network throughput. The instances are of types medium, small, and tiny running Ubuntu 18.04 LTS. The University of Klagenfurt (AAU) [17] provisions virtualized large instances with 12-core AMD Ryzen Threadripper 2920X processors at 3.5GHz and 32GB of memory, and medium with 8-core processor and 16GB of memory running Ubuntu 18.04 LTS. Finally, five NVIDIA Jetson Nano (N/N) running Linux for Tegra (L4T), three Raspberry Pi-3 B+ (RPI3B+), and 32 Raspberry Pi-4 (RPI4) with Raspberry Pi OS complete the testbed.

B. Related Work Comparison

We compare M-RAP with three state-of-the-art methods investigating application placement on Fog and Cloud.

- 1) *Availability-aware placement (AAP)* [9] uses a greedy algorithm to improve the application deadline satisfaction upon failures. For this purpose, it partitions the Fog devices into hierarchical clusters based on network connectivity without considering the resource characteristics. AAP places applications with higher deadlines onto the Cloud upon insufficient Fog devices.
- 2) *Resource-aware placement (RAP)* [15] uses a fractional selectivity model to place services onto the Fog and Cloud based on their requirements. RAP optimizes resource utilization and response time but ignores network connectivity.
- 3) *Joint container placement (JCP)* [16] uses particle swarm optimization to improve the placement rate and deadline satisfaction of monolithic applications in a dynamic Fog and Cloud environment. JCP does not support workflows.
- 4) *Extended M-RAP* uses Cloud instances upon insufficient Fog devices to enable a fair comparison against the previous related methods. After Algorithm 3 sorts and prioritizes the placement of applications with lower deadlines onto the available Fog devices, it places the remaining ones with higher deadlines onto Cloud instances with the highest similarity score and the lowest transmission time to the requesting user to satisfy their resource demands.

VIII. SIMULATION EXPERIMENTS

We evaluated the M-RAP method using the YAFS [32] simulator on an Intel Core^(TM) i7-8650U processor at 1.90GHz and 16GB of RAM, running Ubuntu 18.04 LTS.

TABLE V
TIME-CRITICAL APPLICATION CHARACTERISTICS

Application type	Message size [MB]	Service deadline [ms]
E-commerce [2]	0.01 – 1	50 – 100
Video processing [1]	0.01 – 10	100 – 300
Face detection [4]	0.2 – 14	80 – 100
Virtual reality [3]	0.002 – 50	20 – 50
Natural language processing [5]	3 – 5	100 – 200

TABLE VI
SIMULATION SCENARIOS

Evaluation	Scenarios	App.	Services	Users	App. requests	Service requests
Service placement	SMALL	10	63	26	26	164
	MEDIUM	20	124	60	60	387
	LARGE	30	198	101	101	652
Response time, Deadline satisfaction	D-SMALL	10	63	26	84 429	573 458
	D-MEDIUM	20	124	60	161 597	933 944
	D-LARGE	30	198	101	216 038	1 163 571

A. Workload Simulation

We use the `Gn_Graph` function from the `NetworkX` package to generate workflows compliant with the model in Section III-E. Each request has a size in the range 0.002 MB–50 MB, which generates a service workload of range 100 MI–500 MI according to a representative data stream workflow [1]. Each service has a deadline of range 20 ms–200 ms based on the analysis of five types of time-critical applications (see Table V). We generated the application microservices and their resource requirements (i.e., processing speed, number of cores, memory, and storage size) using a uniform random distribution based on a previous study [9].

B. Fog Simulation

We simulated the Fog as a bidirectional Albert-Barabasi random graph [33] with 100 C^3 devices, as recommended in [9], [34], using the Python `NetworkX` package. We selected 25 devices with the highest betweenness centrality [35] as Fog gateways. The device with the lowest betweenness centrality represents the Cloud data center. We configured the network latency and bandwidth by sending ICMP echo requests and measuring the maximum throughput using the `iPerf3` tool [17] (see Table IV). We simulated the dynamic Fog over 10000s and configured the joining and leaving Fog devices using a uniform random distribution based on range 90–99% availability model [36]. We selected a simulation time interval 20ms representing the lowest service deadline of the simulated time-critical applications (see Table V). We monitor the leaving and joining devices in each time interval and perform incremental multilayered resource partitioning upon Fog infrastructure changes.

C. Simulation Scenarios

We divide the simulation into three parts, shown in Table VI. We split the 10000s simulation in five *simulation periods* (i.e., ΔT_1 – ΔT_5) of 2000s each to simplify the analysis and direct comparison with the AAP method [9].

- 1) *Dynamic service placement* evaluates the Fog placement rate, failure rate, resource wastage, and hop distance in

three scenarios (i.e., SMALL, MEDIUM, LARGE). Each scenario contains randomly generated application sets, the number of services, users, and application requests.

- 2) *Runtime analysis* evaluates the application response time and failure overhead for the three scenarios (D-SMALL, D-MEDIUM, and D-LARGE).
- 3) *Deadline satisfaction* evaluates the application response time in three scenarios (D-SMALL, D-MEDIUM, D-LARGE) in a faulty and reliable Fog.

D. Dynamic Service Placement

We evaluate the dynamic placement regarding Fog placement rate, resource wastage, and hop distance.

1) *Fog Placement Rate*: Fig. 6 shows the placement and failure rates (marked “F” calculated as the ratio between the number of failed services and the total requested services S_μ) on the Fog devices over the 10000s simulation time in the three scenarios.

SMALL: Fig. 6(a) shows that M-RAP achieved an average Fog placement ratio of 0.94 during all simulation periods with only 0.03 failed services. Although AAP placed a similar number of services in the Fog during the ΔT_1 , ΔT_3 , and ΔT_5 periods, the failure rate increased from 0.1 to 0.46, leading to an average Fog placement ratio of 0.60. JCP performed worst with an average placement ratio of 0.23.

MEDIUM: Fig. 6(b) shows that M-RAP outperformed the other methods during all time intervals with an average Fog placement ratio of 0.68. M-RAP imposes only 0.034 failures ratio thanks to its incremental partitioning approach. Although JCP can also execute applications in a dynamic Fog with low failures ratio (0.03), it only achieved an average placement ratio of 0.23. In contrast, AAP and RAP cannot cope with the changing device availability and placed only 0.44, respectively 0.45 services, with an average failure ratio of 0.14 and 0.15.

LARGE: Fig. 6(c) shows that M-RAP and JCP placed the services in the Fog with only 0.02 and 0.01 failed services, respectively. M-RAP achieved a higher average ratio of 0.4, while AAP and RAP placed only 0.29, respectively 0.26 of the services, with average failure rates of 0.11 and 0.07 (increasing from ΔT_1 to ΔT_5).

Summary: M-RAP outperformed the related methods by placing services onto a consolidated set of devices with high resource similarity, leaving powerful devices for other placements. M-RAP considers the device’s availability and introduces low failures. In contrast, AAP and RAP are static greedy methods that deliver higher wastage, lower placement, and higher failure rates since they do not consider network changes. Although JCP exhibited fewer failures, it placed fewer applications by considering them monolithic.

2) *Resource Wastage*: Fig. 7 compares the four methods’ average core, memory, and storage wastage (see section III-G) over a simulation time of 10000s. Figs. 8 and 9 analyze the *idle resources*, indicating the percentage of underutilized cores and memory on each device d_j^t : $\frac{R_{jk}^t - \sum_{A \in AS_t} \sum_{s_i \in S_\mu} \max\{r_{ik}\}}{R_{jk}^t}$, where $k \in 1, 2$.

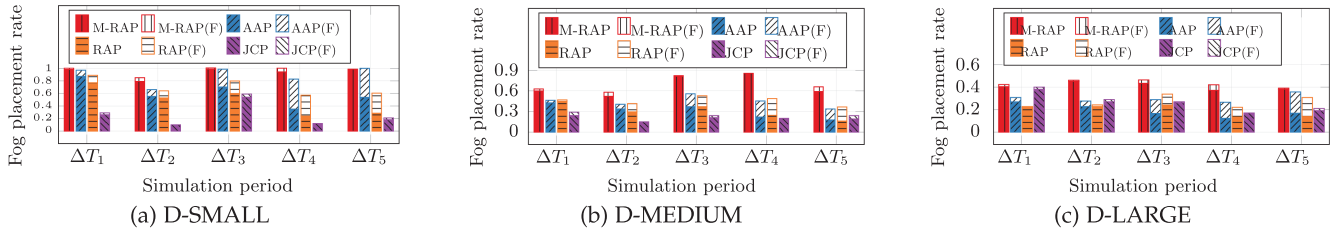


Fig. 6. Fog placement and failure rates during different simulation periods.

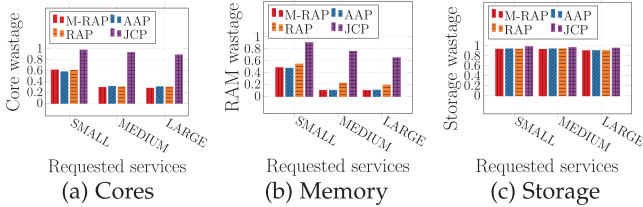


Fig. 7. Simulated average resource wastage.

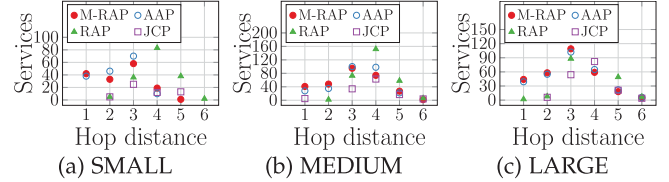


Fig. 10. Simulated average hop distance.

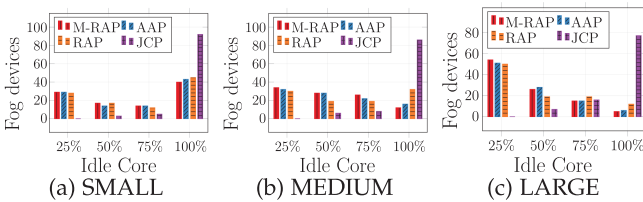


Fig. 8. Simulated average idle cores.

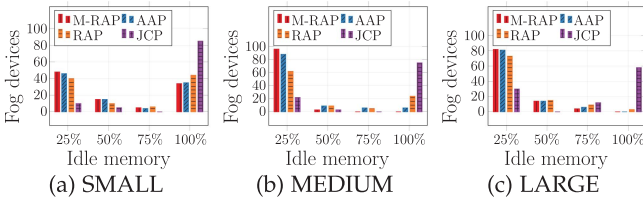


Fig. 9. Simulated average idle memory.

SMALL: M-RAP and RAP provide an average core wastage of 0.60 and a memory wastage of 0.47, while 50% of the Fog devices are idle. AAP delivered a slightly better core wastage of 0.57 and an average memory wastage of 0.54; however, it did not reduce the idle Fog devices below 50%. In contrast, JCP has the highest core wastage of 0.97 and a memory wastage of 0.90 with 95% of idle Fog devices.

MEDIUM: AAP and RAP delivered an average core wastage of 0.31 and 0.3 with 20%, respectively 38% of idle Fog devices. M-RAP outperformed them with a lower average core wastage of 0.29 and 10% idle Fog devices. Once again, JCP performs worst with average core and memory wastage of 0.97 and 85% idle Fog devices.

LARGE: M-RAP has average core wastage of 0.27 and 10% idle Fog devices. AAP and RAP performed equally well (0.3) with insignificantly more idle devices. JCP had the highest wastage of 0.88 and 78% idle Fog devices. We can observe the

same trends for memory wastage and idle memory in all three methods. In contrast, the average storage wastage was very high in all cases, with an average of 0.9 for M-RAP, AAP, and RAP and 0.95 for JCP.

Summary: The core, memory, and storage resources remained vastly underutilized in all methods. M-RAP maximizes the Fog placement rate and consumes more resources, which reduces the core and memory wastage compared to the other methods. We observe no significant differences in storage wastage. M-RAP consolidates resource capacity by placing services on fewer devices with similar requirements, keeping powerful devices available, and reducing resource fragmentation and waste.

3) Hop Distance: Fig. 10 compares the average *hop distance*, indicating the proximity of the hosting devices to the requesting users over the physical network channels during the simulation time of 10000s. One hop distance indicates a service placement at the Fog gateway devices. We compute a hop distance histogram to increase the placement rate at a low hop distance across all application services.

SMALL: Fig. 10(a) shows that M-RAP and AAP placed 42, respectively 38 services at the first hop distance, while RAP and JCP did not manage to place any services. M-RAP and AAP outperformed RAP and JCP by placing 33 and 46 services at the hop distance 2, respectively 56 and 70 services at the hop distance 3. In contrast, RAP and JCP placed 5 services at a hop distance of 2 and 36, respectively 25 at a distance 3.

MEDIUM: Fig. 10(b) shows that M-RAP placed more services near end-users, i.e., 41 services at hop distance 1 and 48 services at hop distance 2. AAP placed 28 services at hop distance 1 and 35 at a hop distance 2. RAP and JCP performed worst and placed few services at hop distances of 1 and 2, and the rest at hop distances of 4 and 5.

LARGE: Fig. 10(c) shows that M-RAP outperformed the other methods by placing 44, 58, respectively 109 services at the first three-hop distances. In comparison, AAP placed slightly fewer services at the first three-hop distances (i.e., 39, 55, and 102).

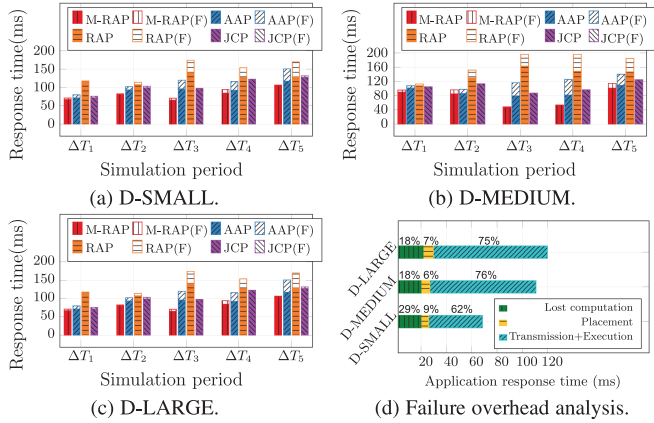


Fig. 11. Application response time in simulation periods.

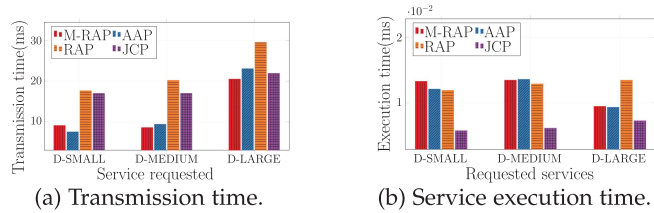


Fig. 12. Simulated transmission and service execution times.

RAP and JCP performed worst by placing 94% of the services at the distances of 3, 4, and 5.

Summary: M-RAP places more services across highly connected Fog devices closer to the users by considering the message transmission time as part of its methodology.

E. Runtime Analysis

We model the total application response time in three aggregated components, following our methodology in [37]:

- 1) *Failure overhead* caused by the leaving Fog devices has two components:
 - a) *lost computation* overhead from a service failure, detected at the end of a time interval, and
 - b) *placement* overhead from executing the incremental resource partitioning and placement algorithms.
- 2) *Transmission and execution times* represent all services' average successful response time after placing the failed ones on new devices.
- 3) *Delay* is the difference between the response time and the deadline in case of a deadline miss.

1) *Response Time:* Fig. 11 shows the average response time and the failure overhead (marked as “F”) in a dynamic Fog. Fig. 12(a) and (b) indicate the simulation’s average transmission and service execution times.

D-SMALL: Fig. 11(a) shows that M-RAP outperformed the other methods for all time intervals with an average response time of 32.42ms and 4.3ms failure overhead. Although AAP performed slightly better for the first three simulation periods, it provided a higher average response time of 92.17ms due to the higher failure placement. RAP and JCP performed worst

with average response times of 128.11ms and 116.6ms due to higher transmission time (see Fig. 12(a)). RAP had a high failure overhead of 41.5ms.

D-MEDIUM: Fig. 11(b) shows that M-RAP reduces the response time and failure overhead to an average of 75ms, respectively 6.6ms due to the lower transmission time (see Fig. 12(a)). The improvements are most evident during the ΔT_5 period due to more devices joining and leaving the network. AAP provides a higher response time with an average of 117.38ms by placing fewer services onto the Fog and ignoring the network changes, leading to higher transmission times (see Fig. 12(a)) and failure overheads. RAP performed worse with an average response time of 168.07ms since it ignores service dependencies and network changes causing higher transmission times (see Fig. 12(a)) and high failure overheads of 29.13ms. Although JCP provides low execution times (see Fig. 12(b)), this negligible improvement does not affect the response time.

D-LARGE: Similar to other scenarios, Fig. 11(c) shows that M-RAP provides the lowest response times with an average of 82.36ms, negligible failure overheads of 4.84ms and lower transmission time (see Fig. 12(a)). Although JCP had an average failure overhead of 3.36ms, it provides a higher average response time of 101ms by placing more services in the Cloud with higher transmission time (see Fig. 12(a)). AAP and RAP performed worse with average response times of 94ms and 124ms, and failure overheads of 19.75ms and 21.76ms. The response time difference is again evident over multiple time intervals, as the static AAP and RAP methods do not consider network changes.

Summary: Fig. 11 shows that M-RAP outperforms the related methods by considering the network interconnections and introducing very low failure overheads. M-RAP places services onto highly connected devices with low transmission time close to users in MEDIUM and LARGE scenarios. In contrast, AAP performs better in the SMALL scenario by applying a greedy algorithm. Fig. 12(b) shows that M-RAP provides a higher execution time by placing the most services in the Fog and outperforming JCP, which places most services in the high-performance Cloud. However, JCP achieves higher response times due to the higher transmission time to the Cloud (see Fig. 12(a)). Finally, AAP and RAP explore static algorithms that do not consider network changes and fail to place all services successfully.

2) *Failure Overhead:* Fig. 11(d) performs an aggregated failure overhead analysis that splits the total average response time of all simulated applications that encountered at least one failed device in three components, following our methodology in [37]. Since most executions exhibited one service failure, the average lost computation is approximately equal to one time interval of 20ms. The placement overhead of executing the low-complexity incremental resource partitioning and placement algorithms is around 8ms (see Section IX).

Table VII summarizes the *severity* [37] of the failure overhead metrics, normalized against the application response time. We restrict the analysis to the executions that exhibited at least one failure to maximize the severity of the failures. Despite the high severity of the lost computation and placement overheads, all applications fulfill their deadlines in the D-SMALL scenario.

TABLE VII
FAILURE OVERHEAD ANALYSIS

Scenario	Lost computation [%]	Placement [%]	Deadline misses [%]	Delay [%]
D-SMALL	29	9	0	0
D-MEDIUM	18	6	9.8	2.2
D-LARGE	18	7	31.4	25

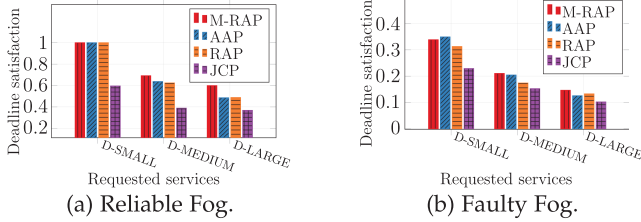


Fig. 13. Simulated average deadline satisfaction.

The number of applications missing their deadlines increases with the scale of the simulated scenario due to the limited number of available Fog devices and the high number of requests. However, our method manages to keep the average delay reasonable (around 38.4ms) in the D-LARGE scenario, representing 25% of the total application response time.

F. Deadline Satisfaction

Fig. 13 analyses the deadline satisfaction of the simulated applications over a simulation time of 10000s.

1) *Reliable Fog Infrastructure*: Fig. 13(a) shows the average deadline satisfaction ratio without faults.

D-SMALL: All three methods fulfilled the deadlines, except JCP placed more applications in the Cloud.

D-MEDIUM: M-RAP outperformed other methods with a slightly higher average deadline satisfaction rate of 0.69 compared to AAP and RAP. JCP performed worst due to its higher response time.

D-LARGE: M-RAP satisfied the deadlines with an average rate of 0.60, while AAP and RAP exhibited a lower 0.48. JCP performed worst with an average rate of 0.39.

Summary: M-RAP has a better deadline satisfaction rate for increasing application requests by placing dependent services across tightly connected Fog devices with lower latency and higher bandwidth within the same network partitions (see Figs. 11 and 12(a)).

2) *Faulty Fog Infrastructure*: We randomly failed Fog devices during the service execution every 20s, such that all devices are not reachable at the end of each simulation period of 2000s. Fig. 13(b) evaluates this faulty Fog's average deadline satisfaction rate.

D-SMALL: M-RAP and AAP fulfilled the deadlines with an average satisfaction rate of 0.34 and 0.35 upon randomly failing Fog devices. RAP performed slightly worse with a cumulative satisfaction rate of 0.31.

D-MEDIUM: All methods fulfilled the deadlines with a lower satisfaction rate than in the D-SMALL scenario. However, the M-RAP performed slightly better than AAP, with an average rate

of 0.21. In contrast, RAP and JCP exhibited a lower deadline satisfaction rate of 0.17, respectively 0.15.

D-LARGE: JCP performed worst and fulfilled deadlines with a low average rate of 0.1, while M-RAP fulfilled their deadlines with a better average satisfaction ratio of 0.14.

Summary: We draw three observations from Fig. 13(b):

- 1) The number of deadline-satisfied applications decreases for fewer Fog devices leading to a high Cloud utilization with higher transmission times.
- 2) RAP and JCP exhibit a lower deadline satisfaction rate than M-RAP and AAP upon faults since they do not consider device failures and provide higher response times.
- 3) M-RAP achieves a higher average deadline satisfaction rate by placing all services across highly connected Fog devices in the same network layer partition. The request routes through another path upon network failures leading to a better deadline satisfaction rate. In contrast, AAP places dependent services across weakly connected devices prone to failures.

IX. REAL TESTBED EXPERIMENTS

A. Implementation

We installed a Docker engine 20.10 on all C^3 testbed devices, which partitions their resources by deploying containerized services isolated from each other based on their resource requirements. The minimal scripts to create and run the containerized services are available in the GitHub repository (<https://github.com/SiNa88/M-RAP>). The lack of interference among the isolated containers hosted by the same device ensures that the service execution times follow the model defined in Section III-F.

We run the M-RAP placement algorithms on a medium C^3 Fog instance type (see Table IV). The multilayer resource partitioning has an average execution time of approximately 3s, executed once for the complete Fog network before the application requests. The incremental multilayer partitioning and service placement algorithms have a significantly faster execution of approximately 3ms and 5ms due to their low complexity with high responsiveness to the incremental changes in the Fog network.

B. Real-World Applications

We validate in this section the simulation results using two real-world applications from the e-commerce and video processing domains, executed on the C^3 testbed (see Fig. 14). We generated ten simultaneous application requests (five for each application) from different users to utilize the testbed and avoid over-utilization entirely. We generated artificial network delays and emulated the distributed locations of user devices using the Linux `tc`, as summarized in Table VIII.

1) *E-Commerce*: E-commerce is a containerized online store application composed of the following components.

Web interface (WI) service provides the graphical interface for interaction with the user;

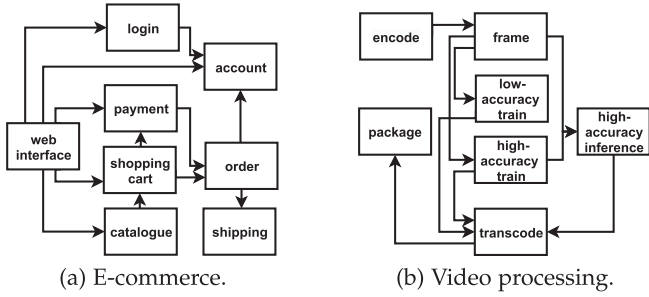


Fig. 14. Real experimental applications.

TABLE VIII
APPLICATION RESOURCE REQUIREMENTS

Application	Processing speed [MI]	Memory [GB]	Storage [GB]	Message [MB]	Deadline [ms]	App. requests
E-commerce	200 – 500	0.1 – 0.5	0.5 – 2	0.01 – 1	100 – 500	5
Video proc.	$10^3 - 10^6$	0.1 – 4	0.1 – 2	0.01 – 10	500 – 2000	5

Login (Log) service authenticates sessions by retrieving usernames and passwords from a MongoDB database;

Catalogue (Cat) service previews product information;

Account (Act) service provides the user profile storing the history of orders;

Shopping cart (Shc) service adds or removes products;

Payment (Pay) service connects the user to the bank payment application and performs the transaction;

Order (Ord) service places the selected product from the stock after inspecting the catalogs;

Shipping (Shp) service manages the shipping information by publishing it to the RabbitMQ message broker.

2) *Video Stream Processing*: Video stream processing is a traffic sign classification application following road safety inspection concerns [1].

Encoding (En) service receives and encodes the high-resolution raw video stream.

Framing (Fr) service uses OpenCV to produce still frames from different video scenes.

Low-accuracy training (LT) service trains a convolutional neural network aiming for a 70% accuracy.

High-accuracy training (HT) service improves the multi-class classification model from newly collected data aiming for an accuracy 90%.

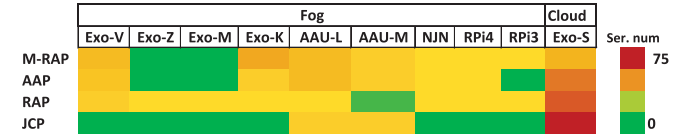
High-accuracy inference (HI) service uses the 90% accurate model to classify the signs in the video frames.

Transcoding (Tr) service converts the video in different resolutions and bitrates and prepares it for delivery by using the ffmpeg software suite.

Packaging (Pa) service delivers the transcoded stream.

C. Experimental Results

Fig. 15 shows the placement heatmap for e-commerce and video processing applications for M-RAP and the three related works on the C^3 testbed. We observe that M-RAP places most services onto Fog devices and very few onto the Cloud instances. AAP and RAP use static greedy methods with higher wastage

Fig. 15. E-commerce and video processing placement heatmap for four related methods in the C^3 testbed.

and lower Fog placement. Finally, JCP places most services into the Cloud instances by considering the applications as monolithic.

E-commerce: Table IX shows that M-RAP provides lower response times by placing all services onto the AAU cluster and Exoscale instances in Klagenfurt, close to the end-users. AAP provides a slightly higher response time by placing several services onto the AAU cluster and the rest onto the higher latency Exoscale instances in Vienna, Klagenfurt, and Sofia. In contrast, RAP places several services onto the AAU cluster and distributes the rest onto the Exoscale instances in Vienna, Zurich, Munich, and Sofia without considering their dependencies, leading to high transmission time. Lastly, JCP places the most services onto Sofia's A1 Cloud data center with a high response time.

Video processing: Table X shows that M-RAP presents the lowest response time again by placing the most services on the Exoscale instances in Vienna and Klagenfurt close to the end-user. However, M-RAP's advantage is less critical due to the high video stream processing workload, which diminishes the role of the transmission time. AAP places the services onto the smaller Exoscale instances, RPi4, and NJN, with higher response times because e-commerce services with an earlier deadline occupy the more powerful instances. RAP places the services onto the different clusters (AAU and Exoscale in Vienna, Munich, and Sofia) without considering their dependencies, leading to higher transmission time. Although JCP places all services onto the Cloud instances, it provides similar response times to AAP and is even lower than RAP since the powerful Cloud instances compensate for the higher transmission time.

D. Result Validation

We validate the correctness of the simulation conducted in Section VIII-E by comparing the response time and the Fog placement in the D-SMALL scenario (see Table VI) with the real testbed measurements for ten e-commerce and video processing requests.

Response time: Fig. 16(a) and (b) reveal that M-RAP has a lower median response time and follows the same trend in both simulated and testbed scenarios. The simulation results show a higher difference in response time than the real testbed due to the higher number of requests and devices and the longer transmission time to the Cloud. M-RAP and AAP placed most applications on the Fog with a lower simulated response time deviation than the real testbed, which placed more applications in the Cloud. M-RAP and AAP placed most applications on the Fog with a lower simulated response time deviation than the real testbed, which placed more applications in the Cloud. M-RAP and AAP placed most applications on the Fog with a lower simulated response time deviation than the real testbed, which placed more applications in the Cloud.

TABLE IX
COMPARATIVE PLACEMENT OF E-COMMERCE SERVICES (WITH THE NUMBER OF PLACED SERVICES IN BRACKETS)

Method	Fog										Cloud			Resource wastage			Response time [s]	Fog placement
	Exo-V	Exo-Z	Exo-M	Exo-K	AAU-L	AAU-M	NJN	RPi4	RPi3	Exo-S	Core	RAM	Stor.					
M-RAP				All(1)	Shc(4), Pay(4), Ord(4)	Act(4), Shp(4)	Cat(4)	WI(4)	Log(4)		0.2	0.47	0.77	0.44	1			
AAP	WI(1), Log(1) Cat(1), Act(1)			All(1)	All(1), WI(1), Log(1) Cat(1), Act(1)	Shc(2), Pay(2) Ord(2), Shp(2)				All(1)	0.55	0.57	0.85	0.58	0.8			
RAP	Act(3), Ord(3)	Pay(2)	Shc(2)				Shp(4)	WI(4)	Log(4)	Act(1), Ord(1) Pay(2), Shc(2), All(1)	0.58	0.57	0.84	0.63	0.58			
JCP					All(1)	All(1)				All(3)	0.83	0.93	0.94	0.64	0.4			

TABLE X
COMPARATIVE PLACEMENT OF VIDEO STREAM PROCESSING SERVICES (WITH THE NUMBER OF PLACED SERVICES IN BRACKETS)

Method	Fog										Cloud			Resource wastage			Response time [s]	Fog placement
	Exo-V	Exo-Z	Exo-M	Exo-K	AAU-L	AAU-M	NJN	RPi4	RPi3	Exo-S	Core	RAM	Stor.					
M-RAP	Tr(3), En(3) HT(3), HI(3)			Fr(3), LT(3) Pa(3)						All(2)	0.2	0.47	0.77	1.75	0.6			
AAP	HT(2), HI(2) Tr(2)						En(2), Fr(2)	LT(2), Pa(2)		All(3)	0.55	0.57	0.85	2.04	0.4			
RAP	En(1), LT(1)	Pa(2)	Fr(2)	En(1), LT(1)	HT(1), HI(1) Tr(1)	Tr(1)				HT(1), HI(1), All(3)	0.58	0.57	0.84	2.47	0.37			
JCP										All(5)	0.83	0.93	0.94	1.9	0			

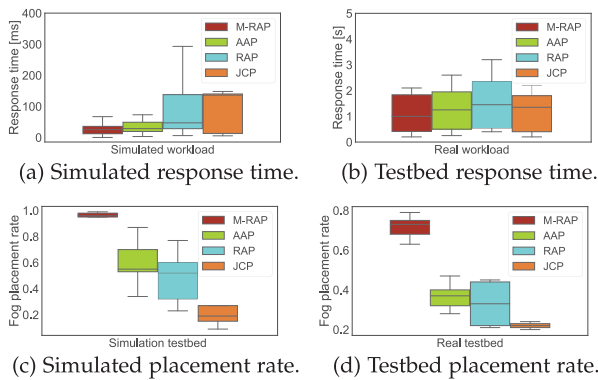


Fig. 16. Simulated versus real testbed results.

response time median for JCP compared to RAP because it placed most of the applications in the A1 Cloud data center in Sofia with a lower transmission time than the AWS or Google data centers in Virginia and Frankfurt.

Fog placement rate: Fig. 16(c) and (d) show that M-RAP provides the highest Fog placement rate, and JCP performs worst in both simulation and real testbed experiments. The simulated Fog placement rate shows higher deviations for all methods except M-RAP due to more requests with different workloads than the real testbed. The higher number of simulated Fog devices leads to a significant placement rate difference between the methods. The fewer Fog devices in the real testbed cause a lower Fog placement rate. The placement rate divergence between the simulation and real testbed indicates that M-RAP, AAP, and RAP reached a high Pearson correlation of 0.87, 0.95, and 0.91, while JCP attains 0.65 because of more variations.

X. CONCLUSION

We introduced a multilayer resource-aware partitioning (M-RAP) method for adaptive application placement in a dynamic Fog infrastructure. M-RAP represents the heterogeneous Fog resources as an incremental multilayer graph and partitions it by considering network connections and resource characteristics. M-RAP constantly updates the multilayer graph and its partitions upon changes in the availability of Fog devices. M-RAP

places the application in two steps. The first step matches the requested application services based on their requirements with feature partitions overlapping in the same network layer partition. The second step places the services on Fog devices closest to the user in the selected partitions. We evaluated M-RAP based on three simulation scenarios considering incremental changes in a dynamic Fog infrastructure during five simulation periods. The results indicate that M-RAP can place 1.6 times as many services, satisfy deadlines for 43% as many application requests, optimize application response time by 58%, and reduce resource wastage by up to 54% compared to state-of-the-art methods. We confirmed the simulation using e-commerce and video processing applications executed on a real testbed comprising eight heterogeneous Cloud and Fog instances distributed over seven geographical locations.

We published our simulation code in the Code Ocean platform (<https://doi.org/10.24433/CO.7045020.v1>) to support the journal reproducibility initiative. In the future, we plan to extend M-RAP to support mobility by:

- 1) assigning and maintaining unique identities of leaving devices, and
- 2) incrementally assigning the mobile devices to the network partitions with the highest proximity and connectivity.

REFERENCES

- [1] N. Mehran, Z. N. Samani, D. Kimovski, and R. Prodan, "Matching-based scheduling of asynchronous data processing workflows on the computing continuum," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2022, pp. 58–70.
- [2] C. Guerrero, I. Lera, and C. Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 6, pp. 2435–2452, 2019.
- [3] W. Zhang, J. Chen, Y. Zhang, and D. Raychaudhuri, "Towards efficient edge cloud augmentation for virtual reality MMOGs," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2017, pp. 1–14.
- [4] E. Cuervo et al., "MAUI: Making smartphones last longer with code offload," in *Proc. Int. Conf. Mobile Syst. Appl. Serv.*, 2010, pp. 49–62.
- [5] Y. Lee, A. Scolari, B.-G. Chun, M. Weimer, and M. Interlandi, "From the edge to the cloud: Model serving in ML.NET," *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 46–53, Dec. 2018.
- [6] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–37, 2019.
- [7] IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing, *IEEE Std 1934–2018*, pp. 1–176, 2018.

- [8] M. Ghobaei-Arani, A. Souri, and A. A. Rahmadian, "Resource management approaches in fog computing: A comprehensive review," *J. Grid Comput.*, vol. 18, no. 1, pp. 1–42, 2019.
- [9] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3641–3651, Apr. 2019.
- [10] H. Sami, A. Mourad, H. Otrouk, and J. Bentahar, "Demand-driven deep reinforcement learning for scalable fog and service placement," *IEEE Trans. Serv. Comput.*, vol. 15, no. 5, pp. 2671–2684, Sep./Oct. 2022.
- [11] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, Jun. 2020.
- [12] J. Edinger, M. Breitbart, N. Gabrisch, D. Schäfer, C. Becker, and A. Rizk, "Decentralized low-latency task scheduling for ad-hoc computing," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2021, pp. 776–785.
- [13] M. Peng, S. Yan, K. Zhang, and C. Wang, "Fog-computing-based radio access networks: Issues and challenges," *IEEE Netw.*, vol. 30, no. 4, pp. 46–53, Jul./Aug. 2016.
- [14] A. Aral and I. Brandic, "Dependency mining for service resilience at the edge," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 228–242.
- [15] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in fog-cloud computing paradigm," in *Proc. IEEE/IFIP Symp. Integr. Netw. Service Manage.*, 2017, pp. 1222–1228.
- [16] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, "Joint container placement and task provisioning in dynamic fog computing," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10 028–10 040, Dec. 2019.
- [17] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, "Cloud, fog or edge: Where to compute?," *IEEE Internet Comput.*, vol. 25, no. 4, pp. 30–36, Jul./Aug. 2021.
- [18] Z. N. Samani, N. Saurabh, and R. Prodan, "Multilayer resource-aware partitioning for fog application placement," in *Proc. IEEE Int. Conf. Fog Edge Comput.*, 2021, pp. 9–18.
- [19] L. Shoostarian, D. Lan, and A. Taherkordi, "A clustering-based approach to efficient resource allocation in fog computing," in *Proc. Int. Symp. Pervasive Syst. Algorithms Netw.*, 2019, pp. 207–224.
- [20] Y. Jie, C. Guo, K.-K. R. Choo, C. Z. Liu, and M. Li, "Game-theoretic resource allocation for fog-based industrial internet of things environment," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3041–3052, Apr. 2020.
- [21] H. P. Sajjad, F. Rahimian, and V. Vlassov, "Smart partitioning of geo-distributed resources to improve cloud network performance," in *Proc. Int. Conf. Cloud Netw.*, 2015, pp. 112–118.
- [22] S. Filiposka, A. Mishev, and K. Gilly, "Community-based allocation and migration strategies for fog computing," in *Proc. IEEE Wirel. Commun. Netw. Conf.*, 2018, pp. 1–6.
- [23] Y. Sun, F. Lin, and H. Xu, "Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II," *Wirel. Pers. Commun.*, vol. 102, no. 2, pp. 1369–1385, 2018.
- [24] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2076–2085.
- [25] A. Yousefpour et al., "FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5080–5096, Jun. 2019.
- [26] Z. Han, H. Tan, X.-Y. Li, S. H.-C. Jiang, Y. Li, and F. C. Lau, "OnDisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2472–2485, Dec. 2019.
- [27] T. X. Tran, P. Pandey, A. Hajisami, and D. Pompili, "Collaborative multi-bitrate video caching and processing in mobile-edge computing networks," in *Proc. IEEE 13th Annu. Conf. Wirel. On-Demand Netw. Syst. Serv.*, 2017, pp. 165–172.
- [28] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, "Intelligent resource allocation in dynamic fog computing environments," in *Proc. IEEE 8th Int. Conf. Cloud Netw.*, 2019, pp. 1–7.
- [29] J. Liu, *Real-Time Systems; Chapter 2: Hard Versus Soft Real-Time Systems*. Englewood Cliffs, NJ, USA: Prentice Hall, 2000.
- [30] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela, "Community structure in time-dependent, multiscale, and multiplex networks," *Science*, vol. 328, no. 5980, pp. 876–878, 2010.
- [31] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Statist. Mechanics Theory Experiment*, vol. 2008, no. 10, 2008, Art. no. P10008.
- [32] I. Lera, C. Guerrero, and C. Juiz, "YAFS: A simulator for iot scenarios in fog computing," *IEEE Access*, vol. 7, pp. 91 745–91 758, 2019.

- [33] A.-L. Barabási, "Scale-free networks: A decade and beyond," *Science*, vol. 325, no. 5939, pp. 412–413, 2009.
- [34] D. Kimovski, H. Ijaz, N. Saurabh, and R. Prodan, "Adaptive nature-inspired fog architecture," in *Proc. IEEE 2nd Int. Conf. Fog Edge Comput.*, 2018, pp. 1–8.
- [35] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, pp. 35–41, 1977.
- [36] P. Pereira, C. Melo, J. Araujo, J. Dantas, V. Santos, and P. Maciel, "Availability model for edge-fog-cloud continuum: An evaluation of an end-to-end infrastructure of intelligent traffic management service," *J. Supercomput.*, vol. 78, no. 3, pp. 4421–4448, 2022.
- [37] R. Prodan and T. Fahringer, "Overhead analysis of scientific workflows in grid environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 3, pp. 378–393, Mar. 2008.



Zahra Najafabadi Samani received the MSc degree in computer architecture from the University of Isfahan, Iran in 2016. She is currently working toward the PhD degree and research assistant since 2019 with the Institute of Information Technology (ITEC), University of Klagenfurt, Austria. Her research interests include resource management and performance optimization in cloud, fog, and edge computing.



Narges Mehran received the MSc degree in computer architecture from the University of Isfahan, Iran in 2016. She is currently working toward the PhD degree and teaching assistant since 2018 with ITEC, University of Klagenfurt, Austria. Her research interests include cloud, fog, edge computing, and future Internet architectures.



Dragi Kimovski received the PhD degree from the Technical University of Sofia, Bulgaria in 2013 and the Habilitation degree from the University of Klagenfurt, Austria, in 2023. Between 2015–2018, he worked as a postdoctoral researcher with the University of Innsbruck, Austria. He is a postdoctoral researcher with ITEC, University of Klagenfurt, Austria. He has co-authored more than 50 international articles in parallel and distributed systems. He participated as a workpackage and scientific coordinator in several European projects.



Shajulin Benedict received the PhD degree from Anna University, Chennai. He is an assistant professor with the Indian Institute of Information Technology, Kottayam, Kerala, India. Afterward, he served as a professor with the St. Xavier's Catholic College of Engineering and a guest professor in Cloud computing with the Technical University of Munich, Germany. He is the director, principal investigator, and representative officer of the AIC-IIITKottayam incubation center for nourishing young entrepreneurs in India. His research interests include the Internet of

Things, performance analysis, cloud scheduling, and edge analytics.



Nishant Saurabh received the PhD degree from the University of Innsbruck, Austria in 2021. He is an assistant professor with the Department of Information and Computing Sciences, Utrecht University, The Netherlands. Previously, he was a postdoctoral researcher with ITEC, University of Klagenfurt, Austria. His research areas include resource management and performance optimization in distributed systems.



Radu Prodan received the PhD degree from the Vienna University of Technology in 2004. He is a professor in distributed systems with ITEC, University of Klagenfurt, Austria. Previously, he was an associate professor with the University of Innsbruck, Austria. His research interests are performance optimization, and resource management tools for parallel and distributed systems. He participated in numerous projects and coordinated the European Union projects ARTICONF and Graph-Massivizer. He has co-authored more than 200 publications and received three IEEE best paper awards.