Mobility-Aware IoT Application Placement in the Cloud – Edge Continuum

Dragi Kimovski, Narges Mehran, Christopher Emanuel Kerth, and Radu Prodan

Abstract—The Edge computing extension of the Cloud services towards the network boundaries raises important placement challenges for IoT applications running in a heterogeneous environment with limited computing capacities. Unfortunately, existing works only partially address this challenge by optimizing a single or aggregate objective (e.g., response time), and not considering the edge devices' mobility and resource constraints. To address this gap, we propose a novel mobility-aware multi-objective IoT application placement (mMAPO) method in the Cloud – Edge Continuum that optimizes completion time, energy consumption, and economic cost as conflicting objectives. mMAPO utilizes a Markov model for predictive analysis of the Edge device mobility and constrains the optimization to devices that do not frequently move through the network. We evaluate the quality of the mMAPO placements using simulation and real-world experimentation on two IoT applications. Compared to related work, mMAPO reduces the economic cost by 28 percent and decreases the completion time by 80 percent while maintaining a stable energy consumption.

Index Terms—Cloud, edge continuum, mobility, application placement, multi-objective optimization, energy consumption, cost

1 Introduction

Internet of Things (IoT) is a disruptive technology that sparked a revolution in terms of connectivity and reachability of the daily used devices. According to Gartner, the number of connected IoT devices will surpass 65 billion by 2025. This will undoubtedly generate enormous quantities of data that require large computational and storage capabilities, currently only offered by massive and centralized data centers. Unfortunately, the latency to reach these data centers can be unacceptably high, especially for time-sensitive IoT applications with strict latency requirements [1].

Recently, *Edge and Fog computing* [2] emerged as extended computing paradigms that partially move low-latency IoT applications from the Cloud closer to the data sources [3], [4]. However, the Edge computing extension of the Cloud services towards the IoT systems raises multiple placement challenges for complex applications modeled as a set of interconnected components [5], such as:

1) *Increased network heterogeneity* that interposes an additional Edge layer between the user and the Cloud;
2) *Limited resource capacity* of Edge (e.g., personal mobile) devices that cannot easily accommodate application requirements, such as processing speed, memory and storage consumption, or communication bandwidth;
3) *High mobility* of Edge devices with severe impact on application reliability and service quality;
4) *Conflicting objective functions* comprising completion time, energy consumption and economic cost;
5) *Heuristic algorithms* to solve this known NP-complete problem considering completion time of the precedence-constrained components as optimization objective [6].

To illustrate the placement challenges, let us consider an IoT application with four sequentially-interconnected components (m_1, m_2, m_3, m_4). The Cloud – Edge infrastructure contains three devices (r_1, r_2, r_3) with different processing capabilities and energy characteristics. The objective is to reduce the application execution time and its energy requirements by identifying proper placements.

Table 1 depicts the execution (including the data transfer) time and energy requirements for each application component on the available devices. We obtain the lowest execution time of 3 by placing m_1, m_2 and m_4 on device r_1, and m_3 on r_3 with an energy consumption of 11. However, we need to place m_1 on r_2 and the remaining components on r_3 to minimize the execution energy to 5, which increases the execution time to 7.5. It is evident that an optimized placement represents a Pareto tradeoff among the two objectives that requires appropriate analysis.

Moreover, the frequent mobility of the devices can negatively influence the application availability. For example, a 10 percent likelihood for the three devices (r_1, r_2, r_3) to leave the network produces a failure probability of 27.8 percent based on the serial reliability model [7]. This further aggravates the application placement problem, especially in heterogeneous environments with hundreds of devices.

To address this problem, we present a novel *mobility-aware multi-objective method for IoT application placement in the Cloud – Edge Continuum (mMAPO)* for IoT applications modeled as a set of lightweight interconnected components [8]. We apply a generation-based multi-objective optimization algorithm that approximates the Pareto set of possible application placements in the Cloud – Edge Continuum using completion time, energy consumption, and economic cost as conflicting criteria. Besides, mMAPO describes the

• *The authors are with the Institute of Information Technology, University of Klagenfurt, 9020 Klagenfurt am Wörthersee, Austria. E-mail: {dragi.kimovski, narges.mehran, radu.prodan}@aau.at, ckerth@edu.aau.at.*

Manuscript received 18 Sept. 2020; revised 29 June 2021; accepted 30 June 2021. Date of publication 2 July 2021; date of current version 9 Dec. 2022.
(Corresponding author: Dragi Kimovski.)
Digital Object Identifier no. 10.1109/TSC.2021.3094322

TABLE 1
Bi-Objective IoT Application Placement Example, With the Optimal Component Placement for Each Objective in Bold

Component	Execution Time			Energy Consumption		
	r_1	r_2	r_3	r_1	r_2	r_3
m_1	**1**	3	2	4	**1**	2
m_2	**0.5**	5	3	5	3	**2**
m_3	2	4	**0.5**	2	4	**1**
m_4	**1**	5	1	1	6	**1**

Edge devices' mobility through a Markov process and improves the Edge resource utilization by placing the applications on devices with lower mobility in the network. Finally, mMAPO implements a low-latency decision-making strategy that selects a single placement solution from the Pareto optimal set based on the application owner demands. mMAPO restricts the application execution to the initial placement and targets adaptation to user mobility as part of future work.

We evaluate the mMAPO application placements using elaborated simulated and real-world Cloud – Edge scenarios against four related methods [9], [10], [11], [12]. We demonstrate the ability of mMAPO to reduce the application completion time by up to 80 percent and decrease the financial cost by 28 percent while maintaining a stable energy consumption. Besides, mMAPO lowers the probability of application failure through mobility characterization of Edge devices.

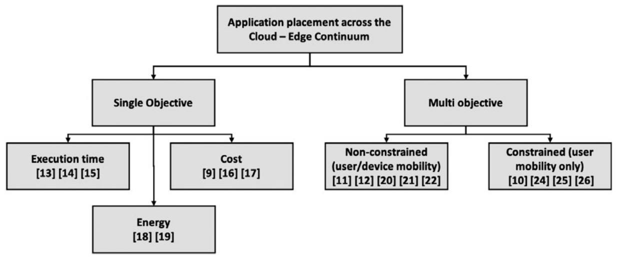
Therefore, the main contributions of this work are:

- A multi-objective IoT application placement model that allocates Cloud – Edge resources to multiple IoT application components based on their characteristics;
- A first-order Markov prediction model for characterization of the mobility of the Edge devices and constraining the multi-objective application placement model;
- Validation on a real Cloud – Edge testbed, extending on related works mostly restricted to synthetic simulations;
- Reduction in completion time by 80 percent and application failure probability by a factor of six compared to four state-of-the-art related methods.

The paper is structured in ten sections. Section 2 surveys the literature on application placement. Section 3 describes our formal model, followed by the mobility-aware placement design in Section 4 and the mMAPO Pareto optimization algorithm in Section 5. Section 6 describes the application case studies, evaluated through experimental simulation in Section 7 and on a real testbed in Section 8. Section 9 empirically evaluates the mobility prediction model, and Section 10 concludes the paper.

2 Related Work

This section discusses related application placement methods across Cloud – Edge Continuum, organized in a taxonomy depicted in Fig. 1.



Application placement across the Cloud – Edge Continuum
Single Objective
Multi objective
Execution time [13] [14] [15]
Cost [9] [16] [17]
Energy [18] [19]
Non-constrained (user/device mobility) [11] [12] [20] [21] [22]
Constrained (user mobility only) [10] [24] [25] [26]

Fig. 1. A taxonomy of application placement in Cloud – Edge.

2.1 Single-Objective Optimization

Existing works for managing Edge devices reduce communication latency, energy consumption or financial costs.

Completion Time. Sun *et al.* [13] investigated a scheduling model that clusters the Edge and Fog devices based on communication latency using a global weighted optimization among and within the clusters to improve task executions. Gupta *et al.* [14] proposed a hierarchical placing of latency-sensitive application components on Edge devices and computationally intensive successors on Clouds. Zhao *et al.* [15] describe a novel dynamic programming approach with support for data stream placement to optimize IoT applications' execution time.

Financial Cost. Aazam *et al.* [16] proposed single-objective management and financial model for Edge resources considering customer behavior and the device heterogeneity to reduce the provider execution cost. Skarlat *et al.* [9] proposed a linear integer programming model to optimize IoT applications placement on Edge devices, considering the execution cost. Similarly, Ni *et al.* [17] utilized a place/transition network to create a list of available Fog, Edge, and Cloud resources and provision them based on the financial costs inquired to the user.

Energy Consumption. Al *et al.* [18] proposed a novel resource allocation approach that utilizes a successive convex approximation to place the IoT applications. Furthermore, Zhang *et al.* [19] proposed an energy-efficient mechanism for partial placement of IoT applications in the Edge through a 5G heterogeneous network.

Research Gap. Existing research searches for placements in a one-dimensional space [9], [15], [17], [18], [19] or reduces the dimensions by weighting the objectives [13], [14]. These methods benefit from lower complexity thanks to the reduced problem space, but may lead to biases by giving nontransparent preference towards an objective.

2.2 Multi-Objective Optimization

Souza *et al.* [11] perform service atomization for optimizing non-functional parameters, such as delay and throughput, by placing the services primarily on Edge devices and applying queuing theory to move the congested ones to the Cloud. Rahbari *et al.* [20] proposed a genetic scheduling algorithm based on a symbiotic organisms that distributes the application components between the Edge and Cloud for optimizing the execution time, energy consumption, and network use. Abdelmoneem *et al.* [21] presented a balance-reduced scheduling algorithm for management of time-critical healthcare applications targeting optimized communication latency and the infrastructure load. Deng *et al.* [22] investigated a nonlinear integer programming optimisation to find deployment schemes that reduce the applications execution costs and meet their average response time. Zhao *et al.* [12] presented edge server selection approach, named GASS, which utilizes a genetic algorithm to place microservice-based applications with sequential combinatorial structure considering the execution time and cost.

Research Gap. Current multi-objective approaches consider the Edge in isolation [11], [12], [20], [22] or model the computing continuum from an Edge perspective [21], [23]. Computing good placements has a high computational overhead due to the complexity of multi-objective optimization, which requires exploring efficient metaheuristics.

2.3 Mobility

Ouyang *et al.* [24] optimized the real-time performance of services in a mobile Edge environment based on a Lyapunov technique to achieve a trade-off between the user-perceived latency and the offloading costs. Wu *et al.* [25] presented a deep learning to predict mobile users' trajectories and take task offloading decisions in real-time based on service quality metrics. De Maio *et al.* [26] presented a genetic meta-heuristic that predicts Edge devices' availability for partially offloading applications based on a trade-off between the user satisfaction and the provider financial profit. Bittencourt *et al.* [10] introduced a resource scheduling strategy using an edge-ward delay-priority to cope with different application classes based high mobility user demands.

Research Gap. The related work focused on predicting user mobility [10], [24], [25], [26] to improve application scalability. However, it fails to address the effect of Edge device mobility on essential application components with respect to the overall execution.

3 Model

This section presents a formal model and a set of definitions essential for this work.

3.1 Application Model

We represent an *IoT application* as a directed acyclic graph $A = (M, D, IN, OUT)$ consisting of:

1) A set of κ lightweight *components*, represented as vertices: $M = \{m_i | i \in \mathbb{N},\ 0 \leq i \leq \kappa\}$;
2) A set of dependencies between the components $D = \{(m_p, m_i, Data_{pi}) | \forall (m_p, m_i) \in M \times M\}$, where $Data_{pi}$ denotes the data transferred from m_p to m_i;
3) A set of input IN and output OUT data requested and produced by the application;
4) A set of *entry* components $m_e \in M$ receiving the input data $Data_e \in IN$ processing request from an IoT device;
5) A set of *exit* components $m_x \in M$ providing the final output data $Data_x \in OUT$ of the application;

We define the predecessor set of m_i as the preceding components executed immediately before m_i

$$pred(m_i) = \{m_p | \forall (m_p, m_i, Data_{pi}) \in D\}.$$

An entry component m_e has no predecessors, i.e.,: $pred(m_e) = \emptyset$. An exit component, in contrast, is no predecessor of any component, i.e., $m_x \notin pred(m_i), \forall m_i \in M$. Our generic graph-based application model has two important special cases covered by our method:

Bag of Tasks. application has no dependencies among its components: $D = \emptyset$.

Monolithic. application has one component and no dependencies, i.e., $M = \{m_1\}$ and $D = \emptyset$.

3.2 Data Model

An IoT device generates data with a stochastic probability, described as a Poisson process. Hence, the probability of observing τ events in a time interval with the total ingress rate of λ_i is: $\Pr(X = \tau) = \frac{{\lambda_i}^{\tau} e^{-\lambda_i}}{\tau!}$. In addition, we consider a *selectivity ratio* ψ_i of a component m_i, defined as the ratio between the egress rate λ_i^{out} and the ingress rate λ_i^{in} of component m_i: $\psi_i = \frac{\lambda_i^{out}}{\lambda_i^{in}}$ [27].

3.3 Resource Model

We define the *Cloud – Edge Continuum* as a group of bounded computing clusters [5], consisting of Edge devices linked to the virtualized instances running in the Cloud.

We consider a set of z heterogeneous Edge and Cloud resources (i.e., containers and virtual machines over physical devices) $RS = \{r_j | j \in \mathbb{N},\ 0 \leq j \leq z\}$. Every resource is a triple $r_j = (\mathtt{CPU}_j, \mathtt{MEM}_j, \mathtt{STOR}_j)$ that describes its number of instructions per second $\mathtt{CPU}_j$, memory size $\mathtt{MEM}_j$, and permanent storage size $\mathtt{STOR}_j$ [28]. The application components run as services deployed within the virtualized resources. Proper execution of a component m_i requires a minimal amount of resources in terms of the number of instructions $\mathtt{INSTR}(m_i)$, processing speed $\mathtt{CPU}(m_i)$, memory $\mathtt{MEM}(m_i)$, and storage $\mathtt{STOR}(m_i)$ requirements identified by the application developers through performance benchmarking [29] prior to the operational deployment.

We model the network of the Cloud – Edge continuum as a set of communication links $\mathcal{L} = \{l_{qj} \mid 0 \leq q, j < z\}$, where a link $l_{qj} = (\mathtt{LAT}_{qj}, \mathtt{BW}_{qj})$ represents the latency $\mathtt{LAT}_{qj}$ and bandwidth $\mathtt{BW}_{qj}$ between the resources r_q and r_j.

We define the *placement* of an IoT application A on a set of Cloud – Edge resources $R \subset RS$ as a function $plc : A \rightarrow R$ that maps each component $m_i \in A$ on a resource $r_j \in R$. The image of the placement function $R = plc(A)$ is the set of resources, in terms of virtualized hardware instances, hosting the running application.

3.4 Mobility Prediction Model

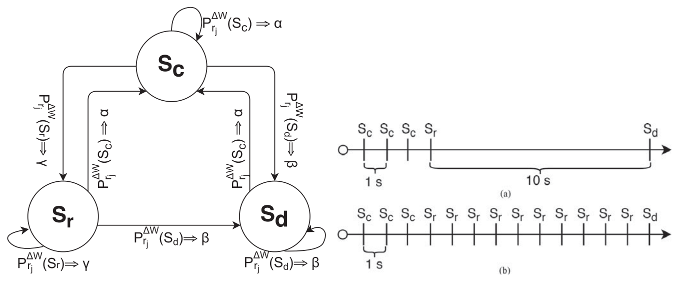
We describe the mobile characteristics of the Edge devices with a Markov model [30] using three states:

Connected. S_c state denotes an Edge device connected to the infrastructure through a higher-level gateway.

Roamed. S_r state implies the movement (roaming) of an Edge device from one higher-level gateway to another.

Disconnected. S_d state denotes that the device left the Cloud – Edge infrastructure and is not accessible.

A transition between states occurs when an Edge device r_j connects to a gateway, roams from one gateway to another higher-level gateway (in another network domain), or disconnects from the network. Fig. 2a represents the Markov state diagram of the three Edge device states $s =$


S_c
S_r
S_d
$P_{r_j}^{\Delta W}(S_c) \Rightarrow \alpha$
$P_{r_j}^{\Delta W}(S_r) \Rightarrow \gamma$
$P_{r_j}^{\Delta W}(S_d) \Rightarrow \beta$
S_c S_c S_c S_r S_d
1 s
10 s
(a)
S_c S_c S_c S_r S_r S_r S_r S_r S_r S_r S_r S_r S_r S_r S_d
(b)

(a) Three-state device mobility. (b) Observation intervals.

Fig. 2. mMAPO Markov model.

$\{S_d, S_c, S_r\}$ and the transition probabilities $P_{r_j}^{\Delta W}(s)$ within a given time window ΔW.

We estimate the transition matrix P by observing the sequence of state transitions at time intervals $t \in \{0, 1, 2, \ldots\}$ within ΔW and assume that P has the right stochastic property, such that the sum of the raw elements equals to one. Thereafter, we normalize the transition matrix for each Edge device r_j based on the three states s within the time window ΔW. Based on the current state of a given Edge device, we transform the matrix into a vector containing only the transition probabilities from the current to the next state used to create transition probability ranges (α, β, γ)

$$\begin{cases} P_{r_j}^{\Delta W}(S_c) : \alpha \Rightarrow [0 : \alpha]; \\ P_{r_j}^{\Delta W}(S_d) : \beta \Rightarrow]\alpha : \alpha + \beta]; \\ P_{r_j}^{\Delta W}(S_r) : \gamma \Rightarrow]\alpha + \beta : 1], \quad \alpha + \beta + \gamma = 1. \end{cases} \tag{1}$$

The Edge device mobility model must consider the regularity of the observation intervals O, as improper observations can lead to biased and incorrect mobility predictions. We denote by $O(X)$ the number of observed occurrences of states $s = \{S_d, S_c, S_r\}$ within a Markov chain X. Observing the Edge devices at non-periodical intervals or only upon state transitions (Fig. 2a) ignores their actual time spent in a given state. To overcome this, it is important to consider both the actual device duration in a particular state and the number of transitions. We, therefore, observe the device state at regular intervals as depicted in Fig. 2b and consider the state duration yielding an improved prediction.

3.5 Optimization Objectives

We consider three objectives for placing an application on the Cloud – Edge continuum: completion time, energy consumption, and economic cost, defined in the next sections.

3.5.1 Completion Time

We define the *execution time* $t_{(m_i,r_j)}$ of a component m_i on a device r_j

$$t_{(m_i,r_j)} = \frac{\mathtt{INSTR}(m_i)}{\mathtt{CPU}_j}. \tag{2}$$

We compute the *completion time* $T_{(m_i,r_j)}$ of a component m_i as the maximum completion time of its predecessors $m_p \in pred(m_i)$, including the data transfer time of $Data_{pi}$, considering the data selectivity ratio ψ_i of m_i

$$T_{(m_i,r_j)} = \begin{cases} t_{(m_e,r_j)}, & pred(m_e) = \emptyset; \\ \max_{m_p \in pred(m_i)} \Big\{ T_{(m_p,r_q)} + \\ \qquad \frac{Data_{pi} \cdot \psi_i}{\mathtt{BW}_{qj}} + \mathtt{LAT}_{qj} \Big\} + t_{(m_i,r_j)}, & pred(m_i) \neq \emptyset. \end{cases} \tag{3}$$

where $r_q = plc(m_p)$. This model eliminates the data transfer between two interdependent components placed on the same resource r_q (i.e., $\mathtt{LAT}_{qq} = 0$ and $\mathtt{BW}_{qq} = \infty$). The completion time of an application A placed on the set $R = plc(A)$ of resources is the latest completion time among its exit components $m_x \in M$

$$T_{(A,R)} = \max_{m_x \in M} \{ T_{(m_x, plc(m_x))} \}. \tag{4}$$

3.5.2 Energy Consumption

The *energy consumption* $E_{(m_i,r_j)}$ of a component m_i executed on resource r_j is the sum of the computation energy $E^p_{(m_i,r_j)}$, the aggregate data communication energy $E^t_{(m_p,m_i)}$ from all predecessors $m_p \in pred(m_i)$, and the static energy $E^s_{(m_i,r_j)}$ of active resources

$$E_{(m_i,r_j)} = E^p_{(m_i,r_j)} + \sum_{m_p \in pred(m_i)} E^t_{(m_p,m_i)} + E^s_{(m_i,r_j)}. \tag{5}$$

The *computation energy* consumed for executing a single component m_i on a resource r_j is

$$E^p_{(m_i,r_j)} = \varrho^p_j \cdot t(m_i, r_j), \tag{6}$$

where ϱ^p_j is the computational power consumption of r_j.

The *communication energy* of r_j's network interface to receive a data size $Data_{pi}$ from a resource $r_q = plc(m_p)$ (including switching and radio communication) is

$$E^t_{(m_p,m_i)} = \varrho^m_j \cdot \frac{Data_{pi} \cdot \psi_i}{\mathtt{BW}_{qj}} + \epsilon_j, \tag{7}$$

where ϱ^m_j is the power consumption of r_j for receiving a data item and ϵ_j is a hardware-related constant [31], [32].

The energy $E_{(A,R)}$ of executing an application A is the total energy consumed by its components

$$E_{(A,R)} = \sum_{m_i \in A \,\wedge\, plc(m_i) = r_j} E_{(m_i,r_j)}. \tag{8}$$

3.5.3 Economic Cost

The *economic cost* $C_{(m_i,r_j)}$ of executing m_i on a device r_j is the total processing, data storage, and communication costs

$$C_{(m_i,r_j)} = t_{(m_i,r_j)} \cdot CP_j + \sum_{m_p \in pred(m_i)} \left\{ Data_{pi} \cdot CS_j + \frac{Data_{pi} \cdot \psi_i}{\mathtt{BW}_{qj}} \cdot CR_j \right\}, \tag{9}$$

where $r_q = plc(m_p)$, CP_j and CR_j are the processing and input data transfer costs of r_j per time unit and CS_j is its storage cost per MB.

The cost $C_{(A,R)}$ of running application A on R Cloud – Edge resources is the total execution cost of its components

$$C_{(A,R)} = \sum_{m_i \in A \,\wedge\, plc(m_i)=r_j} C_{(m_i,r_j)}. \tag{10}$$

3.6 Problem Definition

Multi-objective optimization is typically NP-complete for $o \geq 2$ objective functions $f_1(\vec{\gamma}), f_2(\vec{\gamma}), \ldots, f_o(\vec{\gamma})$, where $\vec{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_\kappa) \in Y$ represents a set of decision variables within a search space Y and κ is the number of space dimensions. The optimization goal is to identify non-dominated solutions in the search space. A solution $\vec{\alpha} \in Y$ *dominates* another solution $\vec{\beta} \in Y$ only if it is better with respect to all objectives: $f_u(\vec{\alpha}) \leq f_u\left(\vec{\beta}\right), \forall u \in [1,o]$, and $\exists v \in [1,o]$ such that $f_v(\vec{\alpha}) < f_v\left(\vec{\beta}\right)$. The resulting set of non-dominated solutions, known as *Pareto optimal set*, represents the tradeoff values among the objective functions. The Pareto optimal set forms the *Pareto frontier* of finite points of tradeoff solutions.

We define a three-dimensional optimization problem ($o = 3$) using the objectives described in Section 3.5. The set of decision variables contains the application components placed onto the Cloud – Edge resources: $M = \{\gamma_i | i \in \mathbb{N},\ 0 \leq i \leq \kappa\}$, where $|M| = \kappa$. Each decision variable γ_i is the placement of one component m_i onto a resource: $\gamma_i = plc(m_i)$. The goal is to find a placement $plc(A)$ for an application A that assigns all its components to the set R of resources that minimizes the three objectives

$$\begin{cases} f_1(T) = \min_{plc(A)=R} T_{(A,R)}; \\ f_2(E) = \min_{plc(A)=R} E_{(A,R)}; \\ f_3(C) = \min_{plc(A)=R} C_{(A,R)}. \end{cases} \tag{11}$$

Searching an optimal placement $plc(A)$ results in a set of solutions, which must satisfy the processing, memory and storage constraints on device $r_j = (\texttt{CPU}_j, \texttt{MEM}_j, \texttt{STOR}_j)$ assigned to each component m_j and the movement probability of a device r_j within a given time window $P_{r_j}^{\Delta W}(s)$

$$\begin{cases} \texttt{CPU}(m_i) < \texttt{CPU}_j; \\ \texttt{MEM}(m_i) < \texttt{MEM}_j; \\ \texttt{STOR}(m_i) < \texttt{STOR}_j; \\ P_{r_j}^{\Delta W}(s) < \texttt{MOB}(m_i), \end{cases} \tag{12}$$

where $\texttt{MOB}(m_i)$ is a mobility constraint defined by the application owner as the highest acceptable state transition probability for a hosting device in the range [0,1]. The component m_i requires a stationary resource r_j if $\texttt{MOB}(m_i) = 0$, and accepts any mobile device if $\texttt{MOB}(m_i) = 1$. The application owner can define higher $\texttt{MOB}(m_i)$ values if it implements fault tolerance recovery mechanisms, such as alternative execution strategies [33]. This is essential, as the mobility prediction model does not implement any recovery strategy in case of incorrect prediction.

4 Architecture

We integrated the mMAPO application placement method and the Edge mobility prediction model over the Carinthian Computing Continuum (C^3) [29] (see Fig. 3). The application owners use a graphical tool or an XML-based language to describe the components, their control, and data flow interactions without exposing the low-level technological details underneath. C^3 registers the available Cloud and Edge resources in its Resource Manager, which maintains a list of JSON files' resource properties. The Resource Analysis module then conducts performance analysis of the registered resources in terms of computing performance, network latency and carbon footprint.

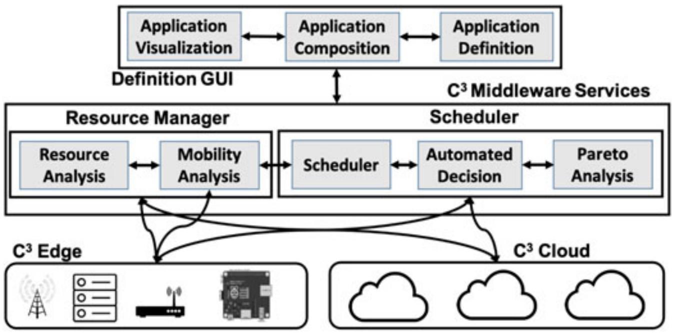


Fig. 3. C^3 infrastructure and mMAPO method integration.

Based on the Resource Manager's underlying resources, the Scheduler places the application using several heuristics, such as the multi-objective optimization algorithm proposed in this paper. The Pareto Analysis module identifies a set of non-dominated Pareto placements on a set of Cloud – Edge resources following a relatively fast evolutionary Non-Dominated Sorting Genetic Algorithm (NSGA-II) [34] that considers the objectives described in Section 3 and the specific mobility characteristics of the Edge devices. To achieve this, it ranks the population according to a fast non-dominated sorting method to prepare elitism and good convergence near the true Pareto optimal set. It inspects if the placements satisfy resource constraints and mobility patterns. An Automated Decision-making module selects an appropriate placement based on a low latency strategy, extending on a simple and computationally efficient a-priori method [35] (see Section 5). Afterward, the Scheduler constructs a single application placement and provides it to the Resource Manager that deploys the components isolated in containers for security reasons.

A monitoring module of the Resource Manager continuously observes the resources and constructs the Markov chain-based mobility prediction.

5 mMAPO Pareto Analysis Algorithm

Algorithm 1 implements the step-wise mMAPO optimized application placement integrated with the Pareto Analysis and Automated Decision modules depicted in Fig. 3. Its input parameters are the application A, the set of Cloud – Edge resources R, the maximal number of generations of $Gen_{\max}$, the population size $|S|$, the mobility constraint coefficient $\texttt{MOB}(m_i)$ and the probability for state transition of the Edge devices within a time window $P_r^{\Delta W}(s)$.

The decision vector $\vec{\gamma} = (plc(m_1), \ldots, plc(m_\kappa))$ is an individual in the genetic population and represents a possible application placement. Its size is equal to the number of components κ of an application A. We first initialize an empty population set (line 1) with random individuals (line 4) and remove those placements that do not meet the apply the application resource and mobility constraints (line 5). Thereafter, we evaluate the remaining random placements based on the three objective functions (lines 6 – 7). Afterwards, we sort the placements using a non-domination sorting algorithm and select the parent individuals for the next generation (line 10). To evolve the next generation, we select two random placements from the sorted parents' population and a two-point crossover (line 14) to create a child placement. We apply a mutation operator with a low probability (line 15) to ensure diversity in the population and discard it if it violates the resource or mobility constraints (line 16). Afterwards, we evaluate the newly generated placement $\vec{\gamma}$ and add it to the population set (line 17). We repeat this process until the algorithm reaches the maximal number of evaluations $Eval$, which is equal to the population size $|S|$ (not considering placements that violate the constraints). Finally, we sort all placements in the population based on non-dominance and select the parent solutions for the next generation (line 21). We repeat the evolutionary process (lines 11 – 22) until reaching a maximal number of generations $|Gen_{\max}|$. We identify the non-dominated placements from the last generation and construct the Pareto optimal set (line 23).

Algorithm 1. mMAPO Multi-Objective Placement Algorithm

```
Input: $A = (M, D, IN, OUT)$                          ▷ IoT application
       $R = (r_1, \ldots, r_z)$                       ▷ Set of Cloud – Edge resources
       $|Gen_{\max}|$                                 ▷ Maximal number of generations
       $|S|$                                          ▷ Population size
       $\texttt{MOB}(m_i)$                            ▷ Mobility constraint coefficient
       $P_r^{\Delta W}(s)$                            ▷ Probability for transition within $\Delta W$
Output: $plc : A \to R$                               ▷ Placement function
1:  $Y \leftarrow \emptyset$                          ▷ Empty population
2:  $i \leftarrow 0$                                  ▷ Generate first individuals
3:  while $i \leq |S|$ do                             ▷ Create new population
4:      $\vec{\gamma} \leftarrow rand(M, D, R)$       ▷ Initial random placements
5:      if constraint($M, R, D, \vec{\gamma}, P_r^{\Delta W}(s), \texttt{MOB}(m_i)$) = True then
6:          $Y \leftarrow Y \cup evaluate(\vec{\gamma})$   ▷ Add placement
7:          $i \leftarrow i + 1$                      ▷ Generate next individual
8:      end if
9:  end while
10: $Y \leftarrow select(Y, |S|)$                     ▷ Select best placements
11: for $gen \in [1, |Gen_{\max}|]$ do                ▷ Iterate through generations
12:     $i \leftarrow 0$
13:     while $i \leq |S|$ do
14:         $\vec{\gamma} \leftarrow crossover(Y)$    ▷ Crossover two placements
15:         $\vec{\gamma} \leftarrow mutation(\vec{\gamma})$   ▷ Mutate placement
16:         if $constraints(M, R, D, \vec{\gamma}, P_r^{\Delta W}(s), \texttt{MOB}(m_i)))$ then
17:             $Y \leftarrow Y \cup evaluate(\vec{\gamma})$   ▷ Add placement
18:             $i \leftarrow i + 1$                  ▷ Generate next individual
19:         end if
20:     end while
21:     $Y \leftarrow select(Y, |S|)$                 ▷ Select best placements
22: end for
23: $Y \leftarrow filter\_dominated(Y)$               ▷ Remove dominated placements
24: $plc(A) \leftarrow decision\_making(Y)$           ▷ Select Pareto placement
25: return $plc(A)$                                   ▷ Return final placement
```

Algorithm 2 receives the application components M, the set of resources R, and the placement $\vec{\gamma}$. It uses the information to remove those individuals that do not meet the $\texttt{CPU}$, $\texttt{MEM}$, $\texttt{STOR}$ *and* $P_r^{\Delta W}(s)$ constraints (lines 5 and 16).

A low-latency decision-making algorithm described in Algorithm 3 selects the preferred placement from the Pareto optimal set by clustering it into priority regions equal to the number of objective functions. The algorithm starts by identifying centroids for each objective function (line 3) and initializes them close to the objective function extremes by considering an objective priority vector $\overrightarrow{OPV} = (x, y, z)$, where x, y, z are priority coefficients for each objective such that $0 \leq x, y, z \leq 1$ and $x + y + z = 1$. Using the centroids, we create clusters for each objective function based on an arithmetic distance (line 4), where each cluster corresponds to an objective priority region. The algorithm finally selects a random placement from an objective priority region based on the decision maker's preference (line 5).

Algorithm 2. mMAPO Placement Constraints Verification

```
1:  function constraints$M, R, D, \vec{\gamma}, P_r^{\Delta W}(s), \texttt{MOB}(m_i)$
2:      $M = (m_1, \ldots, m_\kappa)$                 ▷ Application components
3:      $\vec{\gamma} = (plc(m_1), \ldots, plc(m_\kappa))$
4:      for $i \in [1, |M|]$ do
5:          if $\texttt{CPU}(m_i) > \texttt{CPU}(r_j) \vee \texttt{MEM}(m_i) > \texttt{MEM}(r_j) \vee \texttt{STOR}(m_i) > \texttt{STOR}(r_j) \vee P_{r_j}^{\Delta W}(s) > \texttt{MOB}(m_i)$ then
6:              returnFalse                           ▷ Constraints unfulfilled
7:          end if
8:      end for
9:      returnTrue                                    ▷ Constraints fulfilled
10: end Function
```

Algorithm 3. mMAPO Automated Decision Making

```
1:  function decision_making$Y$
2:      Input: $\overrightarrow{OPV}$                 ▷ Objective priority vector
3:      $IC \leftarrow initiate\_centroids(\overrightarrow{OPV})$
4:      $CP \leftarrow cluster\_pareto(Y, IC)$
5:      return$select\_pareto(CP)$
6:  end Function
```

6 Application Case Studies

We validated the method on two IoT-based medical applications with specific computing and storage requirements, summarized in Table 2. We selected these case studies as they strongly benefit from the Cloud – Edge computing environment, especially in terms of reduced communication and computation latency.

6.1 Insulin Pump

The insulin pump application is a software-controlled system that continuously monitors the blood sugar level using IoT micro-sensors embedded in the patient body [36]. The sensors send the blood sugar information to the resources executing machine learning algorithms to create a model that identifies the patient state variation and computes the proper insulin level upon abnormal state detection. Afterward, the pump

TABLE 2
Resource Requirements per Component

Application	*CPU* [MI]	*MEM* [MB]	*Storage* [MB]	$Data_{pi}$ [Mbit]
Insulin pump	200 – 2000	10 – 60	256 – 1024	0.5 – 4
Mental healthcare	200 – 2000	10 – 50	256 – 512	0.5 – 4

controller receives the information through eight components interacting as in Fig. 4:

- *Compute blood sugar level* of the patient;
- *Compute insulin level* and store it in a remote database;
- *Retrieve patient records* from the database;
- *Review values* of a patient for making proper decisions;
- *Send to doctor* the blood sugar for review of insulin intake;
- *Send review results back* with the proper insulin dose based on the patient's history.
- *Compute pump command* and adjust miniaturized pump pressure to avoid the risk of falling into a coma;
- *Control insulin pump* needle for delivering the correct dose.

The *Retrieve patient records* and *Review values* components must execute in the Cloud in the proximity of the database.

6.2 Mental Healthcare

This application manages near real-time information of patients who suffer from mental disorders [36] in several UK hospitals (Fig. 5). Due to privacy concerns, the patients may not always attend the same clinic and need support through appointments and emergency services:

- *Determine mental state* for a specific patient's record;
- *Decompose the safety concern* for the patient to prevent accidental self-harm and of the general public;
- *Generate record for medical staff* from the current patient disorder stored in a distributed database;
- *View patient history* retrieved from the storage device;
- *Summarize* the record for the physician or the patient;
- *Mental health act* concerning public safety and patient rights, informing relatives to give medicine if serious;
- *Find the closest clinic* and call the emergency or the ambulance services.

The *View patient history* and *Summarize* components must execute in the Cloud where they access secure patient data without moving it over the network.

7 Experimental Simulation

We implemented the mMAPO Pareto analysis algorithm in the jMetal [37] framework and integrated it within the C^3 environment's Scheduler. We created elaborate simulation scenarios using iFogSim [14], which considers the computational and storage characteristics of both the Edge devices and the Cloud virtual machine instances.

7.1 Experimental Design

We evaluated the benefits of mMAPO for application placement compared to the following complementary state-of-

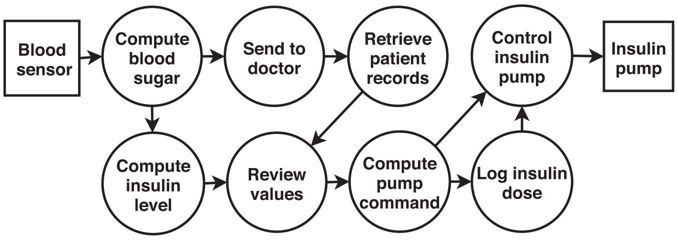


Fig. 4. Insulin pump application.

the-art methods: 1) Fog Service Placement Problem (FSPP) [9] based on linear integer programming model focused on reducing the economic cost and improving resources utilization; 2) Edge-ward delay-priority (EW-DP) [10] that implements a hierarchical best-fit algorithm to cope with users' mobility; and 3) Best-fit Queue (BQ) [11] as a queuing algorithm that uses the Min-Max heuristic [38] to reduce the completion time by giving preference to the Edge devices. We considered completion time, energy consumption, and economic cost for executing a request from the IoT sensors until the final data collection at another device or end-user.

We designed three sets of experiments that consider the characteristics of the IoT applications described in [9] and averaged their results across 1000 completed runs for statistical significance.

We investigate the request failure rate due to the Edge device mobility in Section 9.

Data Size Experiment. (see Section 7.3) investigates the influence on the objectives of $Data_{pi} \in \{0.5, 1, 4, 8\}$ Mbit transferred between the application components, with a fixed application CPU workload of $\texttt{INSTR}(m_i) = 2000$ MI.

CPU Workload Experiment. (see Section 7.4) evaluates the impact of $\texttt{INSTR}(m_i) \in \{250, 500, 1000, 2000, 4000\}$ MI by bounding the data size $Data_{pi} = 4\,\text{Mbit}$.

Component Offloading Ratio. (see Section 7.5) represents the proportion between the number of components κ_o placed on the mobile Edge devices and the total number of placed components κ for given applications A by bounding both the data size $Data_{pi} = 4\,\text{Mbit}$ and the application CPU workload $\texttt{INSTR}(m_i) = 2000$ MI. The higher value for the component offloading ratio represents a higher probability to place a component on a mobile Edge device, which could reduce the availability of the application and lead to failures.

7.2 Simulator Setup

Table 3 summarizes the capabilities of the Cloud – Edge devices divided into three hierarchical categories based on their computing and storage capabilities: 1) Cloud data center; 2)

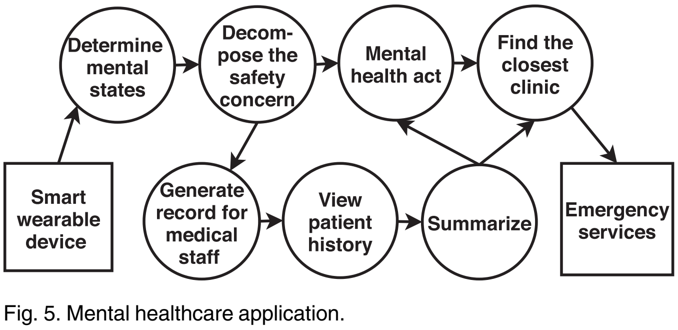


Fig. 5. Mental healthcare application.

TABLE 3
Cloud – Edge Infrastructure Configuration

	Cloud	*Static Edge*		*Mobile Edge*
Resource	Cloud	ISP GW	Cellular BTS	WiFi GW/ME
CPU [MIPS]$\cdot 10^3$	250	65	[10,15]	[2,10]
RAM [GB]	32	16	[8,16]	[0.5,2]
Storage [GB]	512	250	128	16-64
ϱ_j^p [W]	1650	530	[380,410]	[2.50,3.20]
CP_j [¢]	0.03	0.035	[0.04, 0.05]	[0.02,0.04]
CS_j[¢]	10×10^{-7}	15×10^{-6}	[10×10^{-6},20×10^{-6}]	[20×10^{-6}, 30×10^{-6}]
Connectivity	Wired	Wired	WiFi	WiFi, Cellular
Standard	IEEE 802.3a/v	IEEE 802.3a/b	IEEE 802.11a/c	IEEE 802.11a/c/n, 4G/LTE
BW[Mbit/s]	10000	[1000,2000]	[400,1000]	[250, 400]
ϱ_j^m [W]	1300	410	[1.80,2.00]	[1.00,1.50]
CR_j [¢]	3×10^{-6}	35×10^{-7}	[3×10^{-6},5×10^{-6}]	[3×10^{-6}, 5×10^{-6}]

higher-level gateways, including stationary ISP gateways (ISP GW) and cellular Base Transceiver Station (BTS), which are static; and 3) local WiFi gateways (WiFi GW) and mobile Edge (ME) devices, with complex mobility patterns. The Cloud and ISP GW devices have a configuration equivalent with the Intel® Xeon family (i.e., Xeon Platinum 8175) and the WiFi GWs devices with Intel® Core$^{(TM)}$ i7-8550U CPU family. The ME devices are either Raspberry Pi (RPi) single-board computers or mobile phones based on ARM Cortex-A75 architectures with Qualcomm® Kryo$^{(TM)}$ 385 equivalent cores. Ethernet, Wireless LAN, or 4G/LTE network interfaces interconnect the devices.

We simulated a Cloud – Edge environment with a single geographically bounded cluster [5] encompassing twelve ME devices connected to 90 sensors and actuators. A local (ISP) proxy server connects the Edge cluster to a Cloud data center. We assume that the IoT devices and the close sensors/actuators experience a low 1 ms latency, while the latency between every ME and WiFi GW is 10ms. We set the latency between the WiFi GW and ISP GW to 50ms, and between the ISP GW and the Cloud to 100ms, obtained using the Global Ping Statistics in WonderNetwork (https://wondernetwork.com/pings).

7.3 Data Size Results

Figs. 6 and 7 demonstrate that the data transferred between the application components marginalises the placement objectives for both insulin pump and mental healthcare applications. However, we observe several dissimilarities. For different data sizes, mMAPO reduces the average request completion time by up to 70 percent compared to FSPP, which tends to place the application components on the ISP and WiFi GWs farther away from the IoT devices. In contrast, mMAPO identifies application placements on computationally capable Edge devices with low communication latency and low mobility. Although EW-DP and BQ rely more on Edge devices, they apply heuristics or best-fit approaches leading to placements on devices with scarce resources or high mobility patterns. In terms of energy, mMAPO reduces consumption by 17 percent compared with FSPP. Contrarily, mMAPO is less efficient than EW-DP and BQ, which consume 30-50 percent less energy, explained by the tradeoff between the increased computation time per application component on the Edge device and the reduced communication latency required for time-constrained applications. Furthermore, mMAPO reduces the economic cost by up to 79 percent compared to BQ, by 71 percent compared to EW-DP, and by 45 percent compared to FSPP, explained by the lower communication times and improved data locality which reduces the completion times.

7.4 CPU Workload Results

Figs. 8 and 9 demonstrate similar scaling of all methods with the CPU workloads but with substantial performance differences. mMAPO reduces the request completion time by up to 60 percent compared to the three related methods that converge to a local optimum (i.e., EW-DP, BQ) or extensively use high-latency ISP and WiFi GWs, especially for workloads above 1000 MI. Although mMAPO outperforms FSPP by 23 percent for the mental healthcare application, it provides more energy-demanding placements for the insulin pump due the many components concurrently executed on low capacity Edge devices. Besides, EW-DP and BQ consume nearly 55 percent less energy than mMAPO. The higher computation time on the Edge devices leads to higher energy consumption for specific deadline-constrained applications. Finally, mMAPO decreases the economic cost by up to 60 percent compared to EW-DP and 70 percent compared to BQ, thanks to its faster completion time. However, FSPP incurs similar costs to mMAPO by using cheaper Cloud resources than the more expensive Edge, especially for high CPU workloads.

7.5 Component Offloading Ratio

Fig. 10 depicts the average component offloading ratio for mental healthcare and insulin pump applications. We observe that FSPP has a low offloading rate of around 26 percent because it collocates the components on fewer Cloud instances or ISP GWs. On the other side, mMAPO, EW-DP, and BQ

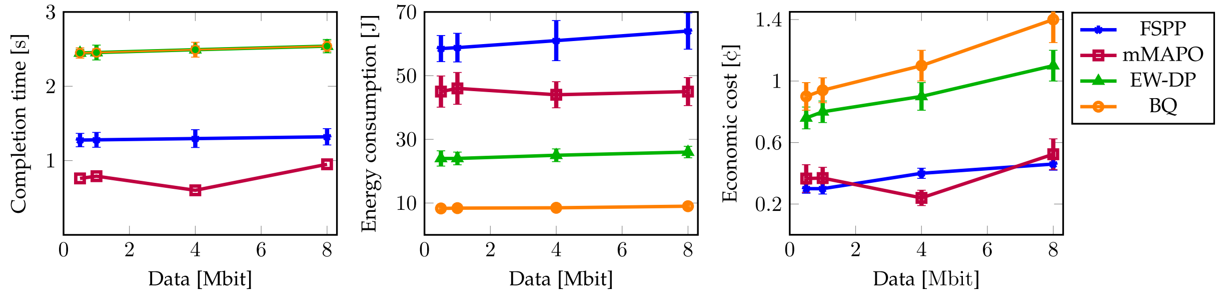


Fig. 6. Insulin pump application completion time, energy consumption, and economic cost for different data size.

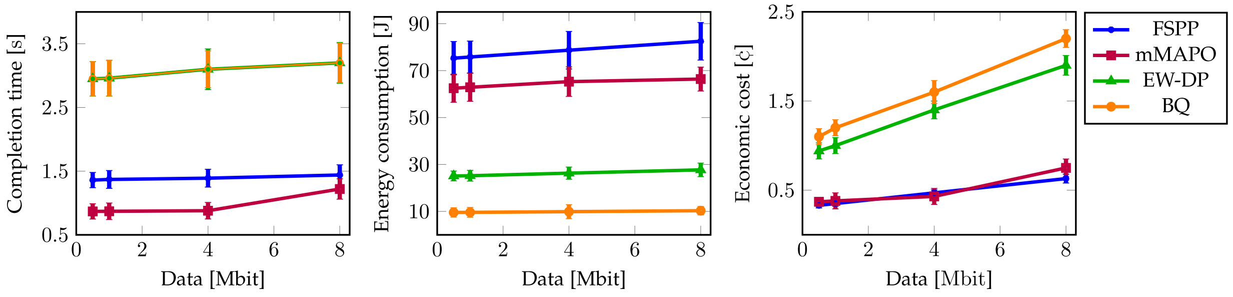


Fig. 7. Mental healthcare application completion time, energy consumption, and economic cost for different data size.

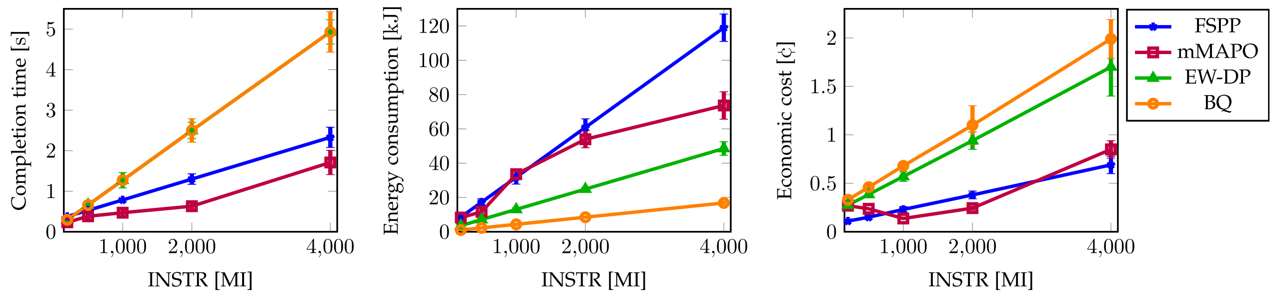


Fig. 8. Insulin pump application completion time, energy consumption, and economic cost for different CPU workload.

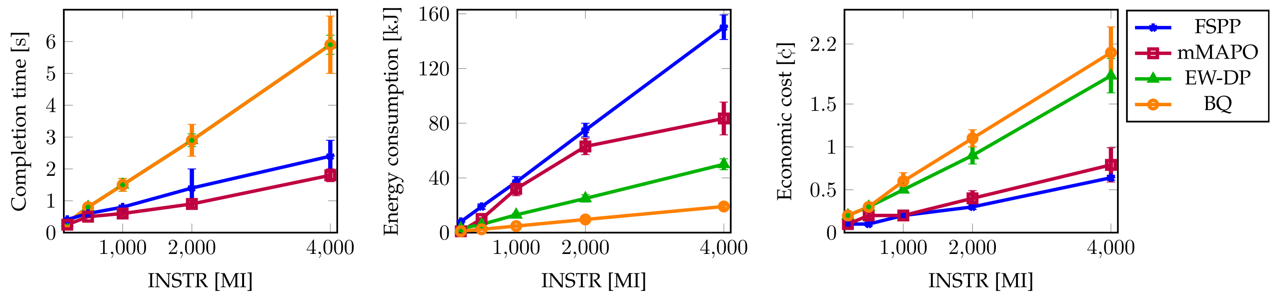


Fig. 9. Mental healthcare completion time, energy consumption and economic cost for different CPU workloads.

achieve high offloading ratios of 81 percent, 83 percent, and respectively 85 percent by giving priority to ME devices. The higher offloading ratio increases the resource utilisation and allows improved distribution of the application components. However, it can lead to lower availability and failures during request processing by placing the components on devices with higher mobility.

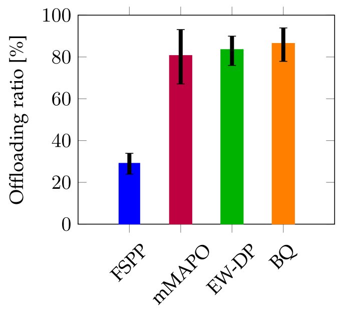


Fig. 10. Component offloading ratio for mental healthcare.

8 Real-World Evaluation

We validated the simulation results by deploying and running the mental healthcare application on a real testbed

TABLE 4
Real Experimental Testbed

	Cloud	*Edge*	
Characteristic	Data center	GW	ME
CPU [MIPS]$\cdot 10^3$	250	65	65
RAM [GB]	16	1	1
Storage [GB]	256	64	64
BW [Mbit/s]	1000	1000	1000

environment. We designed the same two experiments (data size and CPU workload) again as in Section 7.1. We extended comparative state-of-the-art comparison with a new Genetic Algorithm based Server Selection (GASS), which implements a population based heuristic [12] that replicates and distributes the application components to improve the execution time and the fault tolerance.

8.1 Experimental Testbed

We prepared a testbed overlay over the Carinthian Computing Continuum [29], consisting of four Raspberry Pi-3 B+ single-board computers (RPi), three acting as ME, and one acting as GW (see Table 4). We used a virtual machine in a private Cloud with an eight-core Intel® Core$^{(TM)}$ i7-7700 CPU at 3.60 GHz and 15.6 GB of RAM, operated by Ubuntu 16.04 LTS. A dedicated Gigabit Ethernet switch secured using the SSH protocol [39] interconnects the devices.

We installed Raspbian GNU/Linux 9.8 (stretch) and Docker 19.03 on all RPis and deployed a containerized virtualization environment. The IoT devices are close to the MEs in terms of network hops with an average latency of 1ms. The latency between ME and GW is 10ms, and between GW and Cloud is 70ms, measured using the Ping network utility tool over the Internet control message protocol. We employed the Linux Traffic Control utility for managing the network bandwidth and latency between devices by configuring the kernel packet scheduler [40]. We identify the data $Data_{pi}$ transferred between the components through I/O monitoring prior to the experimental deployment. We finally used a plug-in GT-PM-04 power meter to measure the total energy spent by the Edge cluster (connected to the same electric input line) while executing an application.

8.2 Data Size Results

Fig. 11 shows that the completion time and the energy consumption are marginally affected by the communication data sizes, regardless of the placement method. There is an observable difference in the economic cost that increases linearly with the data size. Concerning completion time, mMAPO is six times faster than EW-DP and BQ, five times faster than GASS, and 2.5 times faster than FSPP due to its processing time and communication latency optimization. BQ uses a greedy heuristic that places the application components on a few Edge devices and does not consider higher-level gateways or the Cloud instances, which increase the response time. FSPP collocates multiple components on the same Cloud instance, which becomes a bottleneck for an IoT application with many simultaneous data sensors. EW-DP relies on best-fit algorithms and often omits to identify suitable ME devices. Lastly, although GASS exclusively relies on Edge devices, it performs 10 percent better than EW-DP and BQ, as it distributes the components among the more powerful resources. In terms of energy, mMAPO consumes 17 percent less than EW-DP, 29 percent less than GASS and BQ, and 64 percent less than FSPP. Finally, mMAPO provides 28 percent cheaper placements than EW-DP and BQ, 26 percent cheaper than GASS, and 47 percent more expensive than FSPP, which is the most economic due to the frequent Cloud use.

8.3 CPU Workload Results

mMAPO performs 2.4 times better than FSPP and GASS, and 3.2 times better than EW-DP and BQ for a fixed data size of 4 Mbit and varying CPU workloads of up to 4000 MI, because it considers the execution locality, communication latency to the data sources, and resource availability. GASS achieves similar completion time to FSPP without relying on the expensive cloud instances due to the replication strategy that significantly improves the performance and reliability. Fig. 12 further shows that mMAPO provides up to 46 percent lower energy consumption than FSPP. However, mMAPO is 56 percent worse than EW-DP, 64 percent worse than BQ, and 70 percent worse than GASS, as it identifies trade-offs between performance and power consumption of both the Cloud and the Edge devices. Finally, mMAPO provides 2 percent lower economic costs than the Cloud-bounded FSPP, and between 28 – 34 percent compared to GASS, EW-DP and BQ.

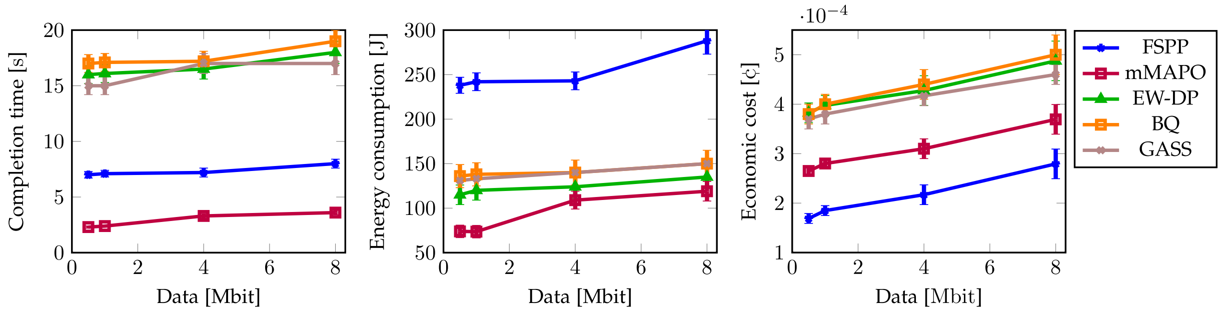


Fig. 11. Mental healthcare application time, energy, and cost for different data sizes.

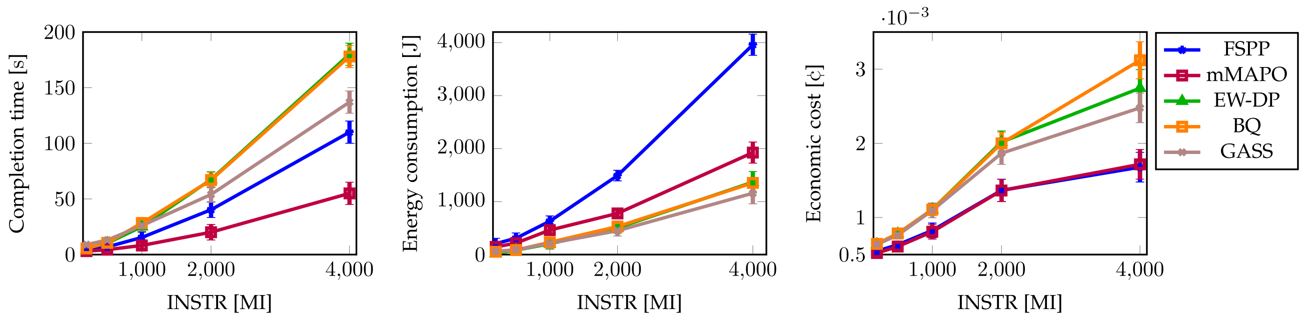


Fig. 12. Mental healthcare application time, energy, and cost for different CPU workloads.

9 Mobility Prediction Evaluation

We present an empirical evaluation of the Markov model-based mobility prediction using real-world traces.

9.1 Experimental Design

We implemented the mobility prediction in Python 3.7.4 using two transition probability matrix calculations.

Incremental Distribution Vector (IDV). creates the initial transition probability matrix by considering a state distribution for all states. It determines the most probable state as a factor of the initial distribution and the product of the probability $P^{\Delta W}$ and the required number of predictions.

Direct Probability Matrix (DPM). uses all observed states in the given time window ΔW to create the initial distribution matrix.

We evaluated a dataset (see Section 9.2) with 316 ME devices and 213 higher-level gateways. We used a fixed observation window of $\Delta W = 6\text{h}$, with an increasing number of observations $O \in \{25, 50, 100, 200\}$, generated by transforming the conditional probabilities to several single transitions for each Edge device. We employed an inverse transform sampling [41] to generate the next state from the probability distribution in Eq. (1). We repeated and averaged single predictions $\{25, 50, 100\}$ times in every experiment.

9.2 Device Mobility Dataset

We utilized the CRAWDAD - ilesansfil/wifidog [42] dataset provided by the Île Sans Fill (ISF) wireless network provider that serves mobile users of Montreal, Canada. The dataset contains 2,177,835 records representing connections among 345 distinct higher-level WiFi GW and 149,861 uniquely identified ME devices. Each record includes, among others, an anonymous user identifier, a node identifier, and the hashed media access control address for the mobile device.

The records spawn over a seven-year period, from 2004 until 2010. We limited our data use to 2009 and 2010, as this period corresponds to the large-scale adoption of smart mobile and IoT devices.

For the transition probability matrix (see Section 3.4), we assumed a dependency between the device mobility and the day's time. We found out that most device movements (roaming) between higher-level gateways occur in the forenoon or afternoon. We, therefore, distributed the selected dataset in four-time windows with a length of six hours each, as follows: 1) ΔW *Morning* ([0, 6[hours); 2) ΔW *Forenoon* ([6, 12[hours); 3) ΔW *Afternoon* ([12, 18[hours); 4) ΔW *Night* ([18, 24] hours). We obtained the initial probability distribution for the Markov chain by weighting the records from the dataset, grouped in similar time observation windows of six hours before applying averaging operations. Besides, we avoided averaging records belonging to different windows ΔW, resulting in unrealistic transition probabilities. Thereafter, we identified the number of connected Edge clusters and the selected 100 most mobile devices connected to the highest number of different clusters per time window. This results in 400 (window, device) tuples corresponding to the dataset in Table 5, used as state observation monitoring data for creating the transition probability matrix P, as well as for generating sample transitions for comparison and evaluation purposes.

9.3 Mobility Prediction Accuracy

mMAPO relies on proper identification of Edge devices with lower mobility. Falsely identifying Edge mobility patterns could discard high-quality placements or consider invalid ones. We, therefore, evaluated the DPM and IDV methods (see Section 9.1) in terms of execution time and prediction accuracy for a single transition. The results in Fig. 13 show that both approaches scale well with the number of observations. However, DPM provides up to 60 percent lower execution times due to the direct calculation of the transition probability matrix. In contrast, IDV utilizes sample vectors and each device's initial states combined with the available observations to create an updated transition probability matrix. Moreover, DPM provides a prediction accuracy for the next state transition of up to 96 percent, with a marginal standard deviation. On the contrary, the observation period affects the accuracy of IDV, which varies between 86 and 96 percent. Lastly, the prediction accuracy is marginally dependent on the number of parallel single-step predictions. Overall, one can assume a slight reduction in accuracy by increasing the number of mobile and unstable Edge devices.

TABLE 5
Dataset Characteristics

Year	*Records*	*Devices*	*Gateways*
2009	12.275	310	200
2010	1.889	193	136

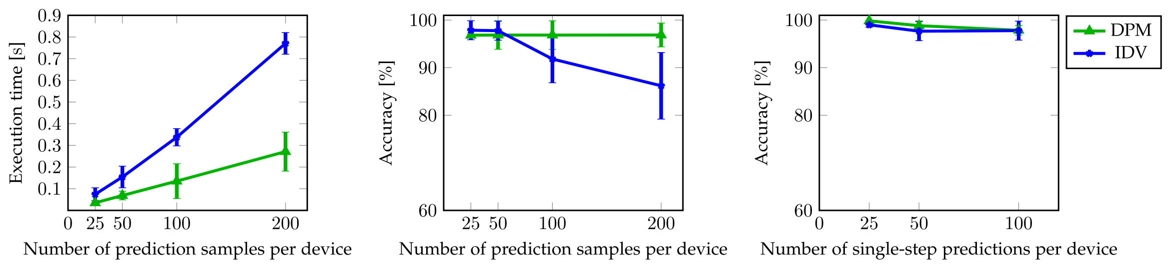


Fig. 13. Experimental evaluation of the Markov chain single-transition mobility prediction model.

9.4 Request Failure Probability

This section evaluates the effect of the mobility on the request failure probability for executing the mental health application. We performed a series-based reliability analysis [43] that identifies the average time spent by the Edge devices in a connected S_c or roamed S_r state in the four time windows ΔW defined in Section 9.2: Morning (MO), Forenoon (FN), Afternoon (AN) and Night (NI). The empirical analysis shows that the average connected or roamed time T for each time window ΔW is 2790s, 2888s, 3322s and 2442s, respectively. We therefore utilize this information to create probability function for a request failure f in time $t = [0\text{s}, 600\text{s}]$ as: $P(f) = e^{-\frac{t}{T}}$. We investigate two mMAPO mobility constraints $\texttt{MOB}(m_i) \in \{0.5, 1\}$ for each application component. We assume for GASS a placement with 50 percent of the components having one replica, which provides a statistically good compromise between fault tolerance and economic cost [44]. Fig. 14 demonstrates that the EW-DP and BQ approaches tend to have around 65 percent higher failure probability, as they primarily rely on mobile devices closer to the Edge of the network. EW-DP particularly places the application components at the Edge to improve the application scalability considering the users' mobility. However, the method does not consider the Edge devices' mobility, which results in a higher failure rate. In contrast, FSPP primarily relies on the Cloud infrastructures with limited use of Edge devices, which results in a relatively low failure probability of around 35 percent. Furthermore, GASS reduces the failure probability to around 25 percent when replicating 50 percent of the components. GASS can reduce the failure probability to only 3 percent when replicating all components once. However this increases the execution price by a factor of two, as the replication does not consider the device mobility and employs more resources for better fault tolerance. Considering the mobility prediction reduces the failure probability of mMAPO to less then 10 percent for $\texttt{MOB}(m_i) = 1$ and further to only 6 percent for $\texttt{MOB}(m_i) = 0.5$, without other financial costs.

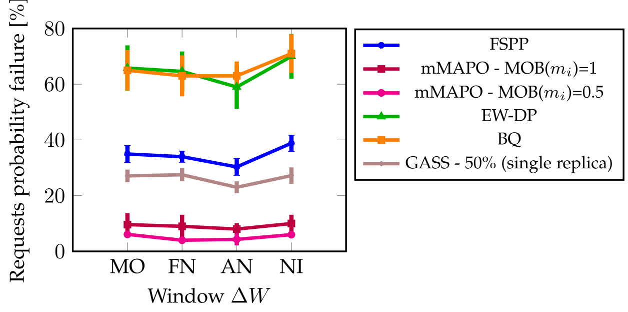


Fig. 14. Average request probability failure for mental healthcare.

Relating the request probability failure with the component offloading rate evaluated in Fig. 10, we conclude that the placement on mobile Edge devices requires fault tolerance techniques in addition to the mobility constraints, as wrong mobility predictions can still lead to request failures of around 10 percent, especially for $\texttt{MOB}(m_i) = 1$.

9.5 Complexity and Quality Analysis

The mMAPO algorithm based on NSGA-II has a complexity of $O(o \cdot S^2)$, where o is the number of objectives and S is the population size [45]. The Markov model for mobility prediction does not influence the complexity, however it reduces the problem space [30].

We investigate the ability of mMAPO to find optimized placements across a large set of Edge devices and Cloud instances in the presence and absence of mobility constraints, using a population of 100 placements. Table 6 compares the mMAPO Pareto analysis algorithm's execution time and the placements' hypervolume [46] for the constrained and non-constrained approaches using a gradually increasing number of placement evaluations. The mobility constrained mMAPO reaches high-quality placements after 5,000 evaluations computed in 266ms on average. Not considering the mobility constraints it requires 10,000 evaluations to reach placements with similar quality, however, with a higher execution time of 332ms. Nevertheless, mMAPO scales well and achieves a maximum execution time of 385ms for the mental healthcare application by deleting placements that do not meet the mobility constraints after each generation.

10 Conclusions and Future Work

We introduced mMAPO, a mobility-aware multi-objective method integrated within the C^3 environment that considers the computation, communication, and mobility aspects for placing and executing IoT applications in the Cloud – Edge Continuum. mMAPO employs a genetic algorithm that optimizes three conflicting objective functions (i.e., completion time, energy consumption, and economic cost), constrained through a Markov chain model characterizing Edge devices' mobility. To solve this problem, mMAPO identifies the interconnections among the application components, analyses the devices' mobility, searches for a Pareto optimal set of tradeoff placements, and selects an appropriate one using an automated decision-making algorithm. We evaluated mMAPO for two medical applications using simulation and real-world experimentation compared against four related methods. Our results show that mMAPO can achieve up to six times lower application completion times, similar energy consumption, and up to 28 percent cheaper execution costs than related

TABLE 6
mMAPO Execution Time and Placement Quality

	Execution Time [s]				Hypervolume			
Evaluations / *Metric*	1000	2500	5000	10000	1000	2500	5000	10000
NC: Non-constrained	0.122	0.181	0.238	0.332	0.567	0.701	0.681	0.715
C: Constrained	0.156	0.204	0.266	0.385	0.658	0.717	0.716	0.725
Improvement: $\left(\frac{C-NC}{NC}\right) \cdot 100$	27.868	13.333	11.764	15.963	15.969	1.295	5.212	1.289

methods. In certain scenarios, mMAPO can even reduce the energy requirements by up to 40 percent. However, it is less efficient than GASS that almost exclusively relies on low-power Edge resources and avoids the Cloud. Moreover, Edge devices' mobility prediction model can reduce the request failure probability by up to 80 percent, even when the component offloading ratio is above 80 percent. The results also show that the Edge infrastructure is more energy-efficient for small applications that do not require high-performance resources. Besides, the communication latency has a larger impact on the completion time than the data size. Lastly, the Edge devices' mobility can significantly impact the execution and deserve proper consideration during the application placement.

In the future, we plan to improve our results by exploring efficient game theoretic methods for placing non-structured applications across the Cloud – Edge continuum.

Acknowledgments

This work was supported by the DataCloud Project, funded by the European Commission under the Horizon 2020 Programme, under Grant 101016835.

References

[1] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog computing in healthcare–A review and discussion," *IEEE Access*, vol. 5, pp. 9206–9222, 2017.
[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. First Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.
[3] M. Hajibaba and S. Gorgin, "A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing," *J. Comput. Inf. Technol.*, vol. 22, no. 2, pp. 69–84, 2014.
[4] M. Villari, M. Fazio, S. Dustdar, O. Rana, D. N. Jha, and R. Ranjan, "Osmosis: The osmotic computing platform for microelements in the cloud, edge, and Internet of Things," *Computer*, vol. 52, no. 8, pp. 14–26, Aug. 2019.
[5] D. Kimovski, H. Ijaz, N. Saurabh, and R. Prodan, "Adaptive nature-inspired fog architecture," in *Proc. IEEE 2nd Int. Conf. Fog and Edge Comput.*, 2018, pp. 1–8.
[6] J. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.
[7] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 193–204.
[8] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using docker and edge computing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 118–123, Sep. 2018.
[9] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware fog service placement," in *Proc. 1st Int. Conf. Fog Edge Comput.*, 2017, pp. 89–96.
[10] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Comput.*, vol. 4, no. 2, pp. 26–35, Mar./Apr. 2017.
[11] V. Souza *et al.* "Towards a proper service placement in combined fog-to-cloud (f2c) architectures," *Future Gener. Comput. Syst.*, vol. 87, pp. 1–15, 2018.
[12] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundancy scheduling for microservice-based applications at the edge," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2020.3013600.
[13] Y. Sun, F. Lin, and H. Xu, "Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II," *Wireless Pers. Commun.*, vol. 102, pp. 1369–1385, 2018.
[14] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw. Pract. Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.
[15] H. Zhao, S. Deng, Z. Liu, Z. Xiang, and J. Yin, "Placement is not enough: Embedding with proactive stream mapping on the heterogenous edge," 2020, *arxiv: 2012.04158*.
[16] M. Aazam and E.-N. Huh, "Dynamic resource provisioning through fog micro datacenter," in *Proc. Int. Conf. Pervasive Comput. Commun. Workshops*, 2015, pp. 105–110.
[17] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu, "Resource allocation strategy in fog computing based on priced timed Petri nets," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1216–1228, Oct. 2017.
[18] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wirel. Commun. Lett.*, vol. 6, no. 3, pp. 398–401, Jun. 2017.
[19] K. Zhang *et al.* "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.
[20] D. Rahbari and M. Nickray, "Scheduling of fog networks with optimized knapsack by symbiotic organisms search," in *Proc. 21st Conf. Open Innovations Assoc.*, 2017, pp. 278–283.
[21] R. M. Abdelmoneem, A. Benslimane, E. Shaaban, S. Abdelhamid, and S. Ghoneim, "A cloud-fog based architecture for IoT applications dedicated to healthcare," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–6.
[22] S. Deng *et al.* "Optimal application deployment in resource constrained distributed edges," *IEEE Trans. Mobile Comput.*, vol. 20, no. 5, pp. 1907–1923, May 2021.
[23] J. Yao and N. Ansari, "QoS-aware fog resource provisioning and mobile device power control in IoT networks," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 1, pp. 167–175, Mar. 2018.
[24] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.
[25] C. Wu, Q. Peng, Y. Xia, and J. Lee, "Mobility-aware tasks offloading in mobile edge computing environment," in *Proc. 7th Int. Symp. Comput.*, 2019, pp. 204–210.
[26] V. De Maio and I. Brandic, "Multi-objective mobile edge provisioning in small cell clouds," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.*, 2019, pp. 127–138.
[27] A. da Silva Veith, M. D. de Assunçao, and L. Lefevre, "Latency-aware placement of data stream analytics on edge computing," in *Proc. Int. Conf. Serv.-Oriented Comput.*, 2018, pp. 215–229.
[28] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, pp. 1–21, 2018.
[29] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, "Cloud, fog or edge: Where to compute?" *IEEE Internet Comput.*, to be published, doi: 10.1109/MIC.2021.3050613.
[30] W.-K. Ching and M. K. Ng, *Markov Chains: Models, Algorithms and Applications*. New York, NY, USA: Springer, 2006.
[31] V. De Maio, R. Prodan, S. Benedict, and G. Kecskemeti, "Modelling energy consumption of network transfers and virtual machine migration," *Future Gener. Comput. Syst.*, vol. 56, pp. 388–406, 2016.
[32] V. De Maio and I. Brandic, "First hop mobile offloading of DAG computations," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2018, pp. 83–92.
[33] V. Sreekanti, C. Wu, S. Chhatrapati, J. E. Gonzalez, J. M. Hellerstein, and J. M. Faleiro, "A fault-tolerance shim for serverless computing," in *Proc. 15th Eur. Conf. Comput. Syst.*, 2020, pp. 1–15.
[34] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[35] D. Kimovski, S. Ristov, R. Mathá, and R. Prodan, "Multi-objective service oriented network provisioning in ultra-scale systems," in *Eur. Conf. Parallel Process.*, 2017, pp. 529–540.
[36] I. Sommerville, *Software Engineering*, 9th ed., Boston, MA, USA: Pearson, 2011.

[37] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Adv. Eng. Softw.*, vol. 42, no. 10, pp. 760–771, 2011.

[38] B. L. Golden, G. Laporte, and É. D. Taillard, "An adaptive memory heuristic for a class of vehicle routing problems with minmax objective," *Comput. Oper. Res.*, vol. 24, no. 5, pp. 445–452, 1997.

[39] T. Ylonen and C. Lonvick, "The secure shell (SSH) transport layer protocol," Cisco Syst., San Jose, CA, USA, Tech. Rep. rfc4253, 2005.

[40] A. Fahs and G. Pierre, "Proximity-aware traffic routing in distributed fog computing platforms," in *Proc. IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2019, pp. 478–487.

[41] S. Olver and A. Townsend, "Fast inverse transform sampling in one and two dimensions," 2013, *arXiv:1307.1223*.

[42] M. Lenczner and A. G. Hoen, "CRAWDAD dataset ilesansfil/wifidog (v. 2015–11-06)," Nov. 2015. [Online]. Available: https://crawdad.org/ilesansfil/wifidog/20151106

[43] M.-S. Chern, "On the computational complexity of reliability redundancy allocation in a series system," *Oper. Res. Lett.*, vol. 11, no. 5, pp. 309–315, 1992.

[44] H. Yu and A. Vahdat, "Minimal replication cost for availability," in *Proc. 21st Annu. Symp. Princ. Distrib. Comput.*, 2002, pp. 98–107.

[45] M. T. Jensen, "Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms," *IEEE Trans. Evol. Comput.*, vol. 7, no. 5, pp. 503–515, Oct. 2003.

[46] L. While, P. Hingston, L. Barone, and S. Huband, "A faster algorithm for calculating hypervolume," *IEEE Trans. Evol. Comput.*, vol. 10, no. 1, pp. 29–38, Feb. 2006.

Dragi Kimovski received the Doctoral degree from the Technical University of Sofia, Bulgaria, in 2013. He was an assistant professor with Ohrid University, North Macedonia, and a senior researcher with the University of Innsbruck, Austria. He is currently a tenure track researcher with the Institute of Information Technology (ITEC), University of Klagenfurt, Austria. He is a work-package leader and scientific coordinator with three Horizon 2020 projects, which include DataCloud, ENTICE, and ASPIDE. He has coauthored more than 50 articles in international conferences and journals. His research interests include parallel and distributed computing, and multiobjective optimization.

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

Christopher Emanuel Kerth received the BSc degree in applied informatics and the MSc degree in distributed systems from the University of Klagenfurt, Austria, in 2015 and 2020, respectively. His research interests include fog and edge computing, and Markov modeling.

Radu Prodan received the PhD degree from the Vienna University of Technology in 2004. He was an associate professor with the University of Innsbruck, Austria, till 2018. He is currently a professor in distributed systems with ITEC, University of Klagenfurt, Austria. He has coauthored more than 200 publications. His research interests include performance, optimization, and resource management tools for parallel and distributed systems. He participated in numerous projects and coordinated, among others, the Horizon 2020 project ARTICONF. He was the recipient of three IEEE best paper awards.