TAROT: Spatio-Temporal Function Placement for Serverless Smart City Applications

Vincenzo De Maio*, David Bermbach†, Ivona Brandic*
*Institute of Information Systems Engineering, Vienna University of Technology, Vienna, Austria, {vincenzo,ivona}@ec.tuwien.ac.at
†TU Berlin & Einstein Center Digital Future, Mobile Cloud Computing Research Group, Berlin, Germany, db@mcc.tu-berlin.de

***Abstract*—Emerging smart city applications (i.e., traffic management, smart tourism) have to (i) process data coming from different IoT devices and (ii) deliver results of data processing to various user devices (e.g., smart vehicles or smartphone) while considering applications' latency constraints. Serverless edge computing has proven to be very effective for latency-aware processing of IoT data, since it allows application developers to define data processing logic in terms of functions which react to data events. However, data processing functions should be dynamically placed and migrated while considering IoT data sources' location and user devices' mobility to minimize end-to-end latency. Unfortunately, current serverless computing solutions do not support mobility-aware placement of functions.**

In this paper, we propose dynamic function placement based on user devices' mobility to address latency requirements of smart city applications. We consider serverless smart city applications, since this computational model allows to model application as a function execution in response to specific events, which makes it suitable for event-driven applications typical of smart city and IoT. First, we identify the parameters affecting end-to-end latency of serverless smart cities' applications. Then, based on our findings, we design TAROT, a latency-aware function placement method based on data-driven mobility predictions. Results show improvements up to 46% for average end-to-end latency in comparison to state-of-the-art solutions.

I. Introduction

Smart city applications (e.g., traffic safety or smart tourism) process data from IoT devices (e.g., traffic lights, closed-circuit cameras, smart glasses) under strict latency constraints and distribute results to mobile end-user devices (e.g., vehicle on-board units or smartphones). While a single sensor value may be small in size, high sampling rates and the vast number of sensors lead to significant data volumes, requiring the use of edge resources [1]. Considering mobile user devices' resource limitations [2], [3], processing IoT data on the Edge (i) reduces resource utilization on user devices and (ii) addresses the latency requirements of smart city applications.

As an execution environment for similar applications, previous work, e.g., [4]–[7], has proposed to follow the serverless paradigm in which application logic is defined as a function execution triggered by specific events. As serverless functions are short-running and stateless, they can be allocated as needed and moved between cloud and edge nodes to adapt to workload changes and application requirements.

An open issue is then the question of when and where to allocate which function. This placement decision should consider application latency, as many smart cities applications have near real-time requirements, but also resource limitations of mobile user devices. Also, function placement should be dynamic in order to adapt to varying resource demand and user mobility. Existing work [7]–[9] either do not consider mobility or applications' latency constraints or do not target smart city applications.

This paper makes the following contributions: (i) we study execution of serverless smart city application as a queuing system, identifying the parameters affecting latency, (ii) we develop TAROT, a data-driven mobility-aware dynamic function placement approach considering device mobility and (iii) design a small-scale prototype of target smart city applications and (iv) develop a simulation of a large-scale smart city infrastructure to evaluate TAROT approach. Simulation is designed based on data collected during execution of the small-scale prototype.

We assume that TAROT is implemented inside a comprehensive serverless computing framework offering (i) dynamic function placement and (ii) data distribution for raw IoT data and processing results. The main goal of TAROT is to perform mobility-aware dynamic function placement on a heterogeneous Edge/Cloud architecture satisfying applications' latency constraints for (i) processing data coming from different IoT sources and (ii) distributing results of processing to different mobile devices. We focus on traffic safety and smart tourism scenarios, as examples of latency-constrained smart city application deployed on a heterogeneous Edge/Cloud infrastructure. We assume a publish-subscribe architecture to distribute IoT data, as typical in many commercial IoT platforms. We consider serverless functions combined in workflows modeling different types of applications.

The paper is structured as follows: first, we describe use cases and foundations (Section II). Next, we analyze data distribution and processing in serverless edge computing as a queueing theory model, to identify parameters that most affect latency of target applications (Section III) and describe TAROT, a mobility-based approach to function placement (Section IV). Afterwards, we describe our experimental setup, both for small-scale deployment and large scale simulation (Section V), before presenting our results (Section VI). Results show improvements in average end-to-end latency (up to 46%) over existing solutions. We discuss related work (Section VII), then conclude the paper in Section VIII.

II. Background

A. Motivating Use Cases

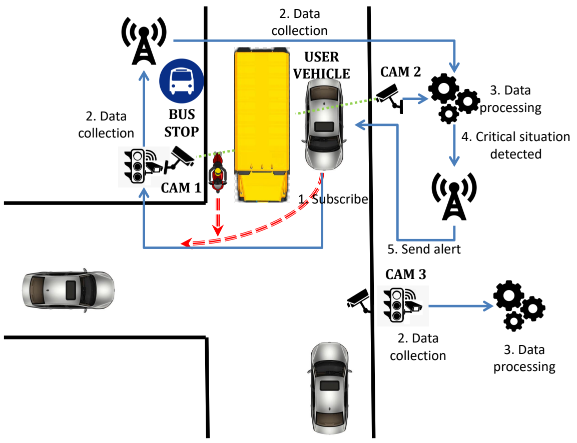


Fig. 1: InTraSafEd5G Use Case.

a) InTraSafEd5G: The InTraSafEd5G project [10], led by our research group at Vienna University of Technology, sends alerts about critical situations happening in a driver's blind spot to improve traffic safety. Figure 1 shows an example with a car approaching an intersection (trajectories marked in red). In Step 1, vehicles subscribes to safety notifications for the intersection, e.g., based on geofences [11]. In parallel, data collected by IoT devices (Step 2) is processed at the edge to identify critical situations, e.g., [12] (Step 3). If data processing identifies a critical situation (in this case, a cyclist hidden from the driver by the bus), it sends this alert to the network (Step 4) which delivers it to all subscribers (Step 5).

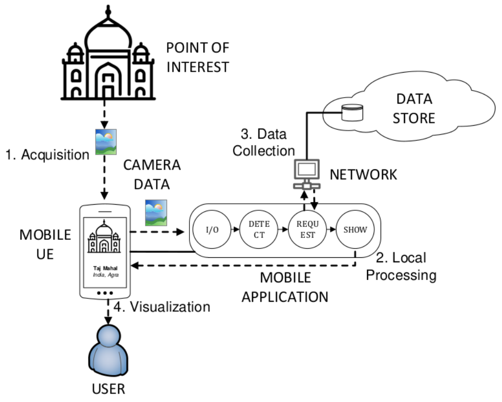


Fig. 2: MobiAR Use Case.

b) MobiAR: MobiAR [13] is a mobile augmented reality application using object detection on data coming from the user device' (UD) camera to recognize different points-of-interest (PoI) in a city and download information and multimedia contents related to it. The whole application flow is described in Figure 2. In step (1), the user points at any direction with UD's camera. Afterwards, in step (2), video streaming data are processed locally, while in step (3), according to object recognition's results, it retrieves data and multimedia (e.g. videos, pictures, documents) associated to the recognized PoI from the networks. Finally, in step (4) MobiAR visualizes the video captured by the camera, augmented with information correlated to the point of interest.

We see that in typical smart city applications, data from a set of different sources need to be collected, processed, and delivered to the UD under strict latency constraints. For each of these steps, different challenges need to be considered: (*i*) subscriptions need to be updated based on the UD's position; (*ii*) data objects are heterogeneous in size and frequency of updates; (*iii*) processing tasks differ in computational requirements, longevity, and priority; (*iv*) connections are unreliable due to the limited resources of IoT devices and mobility of UD; (*v*) applications have different latency constraints. Moreover, these challenges need to be addressed at each intersection (for traffic safety) and at each PoI (for smart tourism). Considering the distribution of traffic lights in urban areas, e.g., in the city of Vienna [14], dynamically placing the data processing logic of InTraSafEd5G and MobiAR is crucial for low latency data processing. Existing approaches, however, focus on content caching [15] or multicast routing [16] and do not consider mobility to improve function placement. We aim to close this gap by providing a latency-aware data distribution and processing considering UDs' mobility.

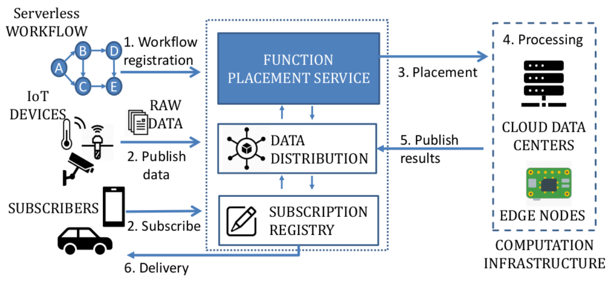


Fig. 3: Overview of Serverless Edge Computing.

B. Serverless Edge Computing

Serverless computing, also known as Function-as-a-Service (FaaS), is an event-driven compute model where application logic is defined by functions [6]. This approach has been shown to be very effective in the IoT context. Currently, there are several FaaS platforms that can also be used at the edge, e.g., Lean OpenWhisk or tinyFaaS [6]. There are, however, some limitations in today's serverless IoT processing:
Function execution: Functions are invoked per event with limited execution time and resources [17]. It may be difficult for developers to predict execution time prior to deployment.
Function placement: In state-of-the-art FaaS frameworks, data is moved to the function, while the other way around would be more efficient in most cases [18]. While FaaS platforms do not support this yet, we assume for this paper that functions can be migrated to data sources or to subscribers as needed [4].

To address the aforementioned challenges, we assume a generic FaaS framework that (i) relies on serverless edge

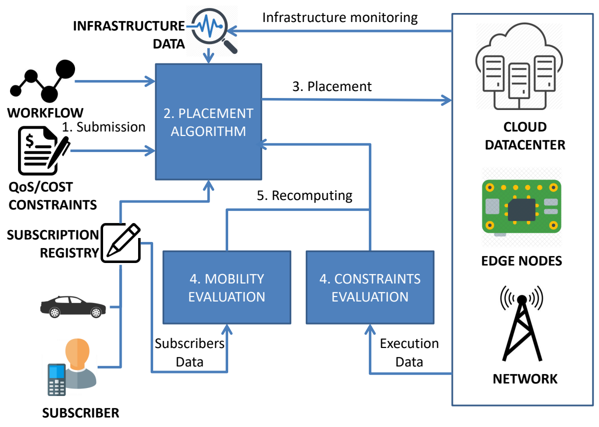


Fig. 4: Overview of the Function Placement Service.

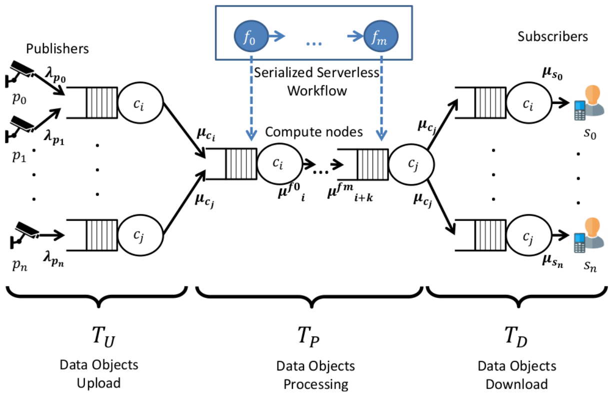


Fig. 5: Serverless Computing Queuing System.

computing for data processing, (ii) ensures in-time delivery of data objects to the function instances processing the data, and (iii) delivers processing results to all subscribed UDs.

Figure 3 gives an overview of QuickFaaS: In Step 1, a serverless workflow is registered to the infrastructure. In Step 2, data coming from IoT devices are published to the network, while users subscribe to their topic(s) of interest. In Step 3, the Function Placement Service (FPS) computes a placement for each workflow function. Once the workflow placement is performed, in Step 4, raw data are processed on the computational infrastructure, and, in Step 5, published to the infrastructure according to subscriptions performed in Step 2. Finally, in Step 6, processing results are delivered to the subscribers. Data exchange between IoT devices, functions, and UDs is handled by a data distribution service implemented through a pub/sub service or a system such as FBase [19].

The FPS is described in Figure 4. We assume that information about available Cloud/Edge nodes is constantly upgraded in the background. First, ready functions are fetched from the FaaS workflow to perform selection of the target. Afterwards, subscriber nodes collect information about each candidate (i.e., network and CPU resources available, future position of UDs) by invoking monitoring services. Based on collected data, each node selects its favorite target and vote for it. Finally, a candidate is selected as target for deploying the functions. In this paper, we describe TAROT as function placement method.

For IoT data distribution, there are protocols such as CoAPor MQTTand service offerings such as Amazon IoT (https://aws.amazon.com/iot/) or Google Cloud Pub/Sub (https://cloud.google.com/pubsub/docs/overview). Most of these follow the pub/sub paradigm since the push delivery of data only to interested parties is a perfect fit for many IoT applications [20]. Consequently, we will assume pub/sub-based data distribution.

III. Serverless Edge Processing Analysis

Data coming from a publisher node $p \in \mathcal{P}$, where $\mathcal{P}$ is the set of publisher nodes, is streamed as a set of *data objects* [21]. Data objects are processed by one or more compute nodes (cloud or edge) $c \in \mathcal{C}$, being $\mathcal{C}$ the set of compute nodes. Processing logic is described by *serverless workflows* W, composed of several interdependent *functions* modeling different data processing stages. Serverless workflows are modeled as directed acyclic graphs (DAGs) where each node represents a function and each edge models precedence relations between functions, i.e., a *function transition*. Once processing is completed, data objects are delivered to all subscribers $s \in \mathcal{S}$ subscribed to the results of W.

Edge nodes are deployed following the smart traffic lights distribution in [14]. Data objects share the infrastructure's network and computational resources which is modeled as the queuing system in Figure 5. Based on the queuing model, we define the *Average workflow latency* T as the average time for each data object to be uploaded, processed, and delivered. Table I summarizes our model notation.

In Figure 5, we define the average workflow latency T as the sum of three components: the *upload time* T_U, i.e., the latency for uploading data objects from publishers to compute nodes, the *processing time* T_P, i.e., the latency for processing data objects in compute nodes, and the *download time* T_D, i.e., the latency for downloading processed data objects from compute nodes to subscribers.

1) Upload Time: Each publisher forwards data objects of different sizes and number in a specific time interval. We assume that each publisher $p_i \in \mathcal{P}$ produces data objects according to a Poisson Point Process with an average rate $\lambda(p_i)$. The average size in bytes for each data object forwarded by p_i is modeled by a random exponential variable with average $\ell(p_i)$. Transmission of data objects is done via the network infrastructure, subject to capacity limits on the communication channel between two nodes. Let $\mathcal{N} \overset{\text{def}}{=} \mathcal{P} \cup \mathcal{C} \cup \mathcal{S}$. The capacity of the channel between $n_i, n_j \in \mathcal{N}$, $r(n_i, n_j)$ is defined by the network bandwidth available in the channel between n_i and n_j, $r(n_i, n_j)$. *Average upload latency* from p to c is

$$\mu_{pc} = \frac{\lambda(p)\ell(p)}{r(p,c)} \tau_{p,c}, \quad (1)$$

where $\tau_{p,c}$ is a binary variable which is set to 1 if p transmits data to c, 0 otherwise. We assume that $\tau_{p,c} = 1$ for all c in the cell of p. Let $\mathcal{A}(c)$ be the set of $p \in \mathcal{P}$ associated to c. $\mathcal{A}(c)$ depends on nodes' $n \in \mathcal{P} \cup \mathcal{S}$ coordinates, $\texttt{coords}(n)$. We define the average network utilization of c, $\mathcal{U}(c)$, as

$$\mathcal{U}(c) = \sum_{p \in \mathcal{A}(c)} \mu_{pc}. \tag{2}$$

Communication channels between publishers and compute nodes are shared in a round-robin fashion [22]. Since $\ell(p)$ follows an exponential distribution and the data rate is given, the *service time* for each p (i.e., the time required to deliver each data entry) follows an exponential distribution [23]. We then define the *average transfer time* from p to c as

$$t_{pc} = \frac{\ell(p)}{r(p,c)}. \tag{3}$$

As a result, the communication between publishers and compute nodes realizes an M/M/1 processor sharing queue, which according to [22] is a feasible model for emulating practical data transmission since (i) each pair (p, c) has different $r(p, c)$ according to the network capacity and geographical distance between nodes and (ii) resources of each c are shared fairly across publishers. The service time of the M/M/1 queue is equal to the transfer time t_{pc}. For analytical tractability, we assume that the system is stable, i.e., that $\mathcal{U}(c) < 1 \ \forall c \in \mathcal{C}$, namely, that all publishers assigned to c do not overload the communication channel of c. The average time for sending data objects from p to c, μ_{pc}, is the sum of the delivery and the waiting time for each data object. The delivery time $t_d(p, c)$ depends on the $r(p, c)$ and $\mathcal{U}(c)$ since the channel is shared among all $p \in \mathcal{A}(c)$ and is defined as

$$t_d(p,c) = \frac{\ell(p)}{r(p,c)(1-\mathcal{U}(c))}. \tag{4}$$

Given the M/M/1 processor sharing queue at each compute node, the average waiting time for data objects generated by p is equal to the difference between delivery and service time, which is equal to t_{pc}. Therefore, the waiting time w_{pc} from p to compute node c is equal to

$$w_{pc} = t_d(p,c) - t_{pc} = \frac{\mathcal{U}(c)\ell(p)}{r(p,c)(1-\mathcal{U}(c))}. \tag{5}$$

We then define the average upload latency for each $p \in \mathcal{P}$ as

$$\hat{T}_U(p,c) = w_{pc} + t_d(p,c) = \frac{\ell(p)(\mathcal{U}(c)+1)}{r(p,c)(1-\mathcal{U}(c))}. \tag{6}$$

From the definition of T_U follows

$$T_U = \sum_{c \in \mathcal{C}} \sum_{p \in \mathcal{A}_p(c)} \hat{T}_U(p,c). \tag{7}$$

2) Compute Latency: Processing of data objects is achieved by executing user-defined FaaS workflows on compute nodes. Since workflows are submitted to the infrastructure at deploy-time, we assume their structure is already known to the placement service. Therefore, from now on, we use the term "workflow" to identify both the DAG and its serialized version, obtained through a topological sorting of workflow W. We define a serialized serverless workflow $W_i = \langle F_i, T_i, topic_i^{in}, topic_i^{out} \rangle$, where F_i are W_i's functions, T_i are the transitions between functions, $topic_i^{in}$ is the list of topics accepted by workflow W_i, and $topic_i^{out}$ the topics under which W_i results are published.

We study the processing of serverless workflows as a deterministic queuing network such that (i) for each function $f_j \in F_i$, $j \in [0, |F_i|]$ we have a queue and (ii) for each transition between f_j, f_k we have an edge connecting the two queues of f_j and f_k. For all W_i, we assume that the computational load (i.e., computing size) of applying a function $f \in F_i$ to data objects forwarded from n, $\texttt{MI}(f, n)$, defined in millions of instructions (MI), follows an exponential distribution [22]. Functions must be deployed on a compute node $c \in \mathcal{C}$ to be executed. The actual deployment of each function f of W_i is determined by the *placement vector* Π_i, such that $\Pi_{i,j} = c$, with $j \in [0, |F_i|]$ if f_j is placed on node $c \in \mathcal{C}$. Since computational capacity $\texttt{MIPS}(c)$ of each $c \in \mathcal{C}$, defined in millions of instructions per seconds (MIPS), is fixed, we define the service time $\hat{s}$ for function f_j on data objects from node n as

$$\hat{s}(f_j) = \frac{\texttt{MI}(f_j, n)}{\texttt{MIPS}(\Pi_{i,j})}. \tag{8}$$

The average processing time of data objects coming from a node n applying function f on node c is defined as

$$\hat{p}(f_i, n_j) = \frac{\lambda(n_j)\texttt{MI}(f_i)}{\texttt{MIPS}(n_j)} \phi_{i,j}. \tag{9}$$

Where $\hat{x}_{ij}$ is a binary variable that is equal to 1 if f_j is deployed on n_j. Values of $\hat{x}_{ij}$ depend on the placement vector Π_i. Let $\psi(c)$ be the set of functions deployed on c. Computational use of node c, $\mathcal{U}^c(c)$, with $c_i \in \mathcal{C}$, is then

$$\mathcal{U}^c(c) = \sum_{f \in \psi(c)} \hat{p}(f, c). \tag{10}$$

Processing of serverless workflows can be modeled as a network of M/M/1 queues, as follows from the Burke theorem. From the properties of M/M/1 queues, we define the waiting time for function f on node c, $w_{f,c}^c$ as

$$w_{f,c}^c = \frac{\mathcal{U}^c(c)\texttt{MI}(f,c)}{\texttt{MIPS}(c)(1-\mathcal{U}^c(c))} \tag{11}$$

and the computing time for function f on node c as

$$t_c(f,c) = \frac{\texttt{MI}(f)}{\texttt{MIPS}(c)(1-\mathcal{U}^c(c))}. \tag{12}$$

The average processing latency for f on node c is then

$$\hat{P}(f,c) = w_{f,c}^c + t_c(f,c). \tag{13}$$

As a consequence, the average processing latency T_p is:

$$T_p = \sum_{j \in [0, |F_i|]} \hat{P}(f_j, \Pi_{i,j}). \tag{14}$$

3) Download Time: Results of workflow W_i are then transferred to all subscribers registered to $topic_i^{out}$, namely $\Theta(\mathcal{S}, topic_i^{out})$. Let $c = \Pi_{|F_i|}$ and $\lambda(c) = \hat{s}(f_j)$. We assume that the size of data objects, which are a result of W_i, $\ell(c)$, follows an exponential distribution. The average download latency for each $s \in \Theta(\mathcal{S}, topic_i^{out})$ is then

$$\hat{T}_D(c,s) = w_{cs} + t_d(c,s) = \frac{\ell(c)(\mathcal{U}(s)+1)}{r(c,s)(1-\mathcal{U}(s))}. \tag{15}$$

where definitions of w_{cs}, $t_d(c,s)$ and $\mathcal{U}(s)$ follow respectively from Equations 5, 4, and 2. The average download time is then

$$T_D = \sum_{s \in \Theta(\mathcal{S}, topic_i^{out})} \hat{T}_D(\Pi_{|F_i|}, s). \tag{16}$$

Finally, we define the average workflow latency as

$$T = T_U + T_P + T_D. \tag{17}$$

We observe that T is influenced mainly by the average data object size ℓ, the arrival rate for data object λ, the network channel capacity r and the network and CPU utilization of each node, respectively $\mathcal{U}$ and $\mathcal{U}^c$. Since ℓ, λ and r depend on the application or infrastructure setup, we focus on the management of $\mathcal{U}$ and $\mathcal{U}^c$ to reduce latency.

Symbol	Description
$\mathcal{N}$	Set of network nodes belonging to serverless infrastructure
$\mathcal{P}, \mathcal{C}, \mathcal{S}$	Sets of publisher, compute and subscriber nodes
$\Theta(\mathcal{S}, t)$	Set of subscribers subscribed to topic t
$\Theta(\mathcal{P}, t)$	Set of publishers that publish data objects under topic t
T	Average Workflow Latency
T_U	Average Upload Time
T_P	Average Processing Time
T_D	Average Download Time
$\ell(n)$	Average size of data objects from $n \in \mathcal{N}$
$\lambda(n)$	Arrival rate of data objects from $n \in \mathcal{N}$
$\mathcal{A}(c)$	Coverage of a node $c \in \mathcal{C}$
$\mathtt{coords}(n)$	Coordinates of node n
$\mathcal{U}(c)$	Network utilization of a node $c \in \mathcal{C}$
$\mathcal{U}^c(c)$	Computational use of a node $c \in \mathcal{C}$
$r(n_i, n_j)$	Network channel capacity between nodes $n_i, n_j \in \mathcal{N}$
$\mathtt{MIPS}(c)$	Computational capacity of node $c \in \mathcal{C}$
$\mathtt{MI}(f)$	Average computational load for function f
W_i	A serialized FaaS workflow
F_i	Set of functions in workflow W_i
T_i	Set of transitions between the functions of W_i
$topic_i^{in,out}$	Topics to which W_i subscribes/publishes
Π_i	Placement vector for W_i
$\Pi_{i,j}$	Placement node for function f_j, $j \in [0, \lvert F_i \rvert[$
$\tau_{i,j}$	1 if publisher i communicates with compute node j, 0 otherwise
$\phi_{i,j}$	1 if function i is deployed on compute node j, 0 otherwise
w_{ij}	Waiting time between nodes $n_i, n_j \in \mathcal{N}$
$t_d(n_i, n_j)$	Delivery time between nodes $n_i, n_j \in \mathcal{N}$
μ_{ij}	Average network latency between nodes $n_i, n_j \in \mathcal{N}$

TABLE I: Table of Notation.

IV. TAROT APPROACH

In this section, we describe TAROT (predic**T**ion-based F**A**aS wo**R**kfl**O**w placemen**T**). TAROT components are the *placement* service, which performs the placement of functions, and the *prediction* service supporting it.

A. FaaS Workflow Execution

In FaaS systems, functions/workflows are deployed on the target infrastructure and executed on-demand following a trigger (i.e., incoming data from specific sources). After invocation, they produce a results and then terminate. Since they are deployed beforehand, information such as workflow structure are known before execution. Also, to prevent over-utilization of resources, each function f_k, $k \in [0, |F_i|]$ is given a deadline f_k^d, which limits the time f_k can run. FaaS workflow execution is described in Algorithm 1.

First, knowing the structure of W allows to pre-compute an order of execution for functions of W, which is done by the function `fetchFunctions`. In lines 4-10, each f_k function is deployed following the order defined by `fetchFunctions`, on the target node defined by the placement vector Π_i, $\Pi_{i,k}$. Since workflows are executed on-demand, computing ideal placement each time a function is invoked might lead to violation of workflows' constraints. Therefore, Π_i is computed at regular time intervals, i.e., each uT seconds. Since we know in advance each function deadline f_k^d, we start computation of placement for next function f_{k+1} if we see that execution of f_k could exceed uT. This is modeled by the condition in line 6, where we consider also the $coldStart$ time, i.e., the time required to allocate function on $\Pi_{i,k}$.

Algorithm 1 FaaS Workflow Execution

```
1: function WF − EXEC($\mathcal{C}, W_i, \Pi_i, uT$)
2:     $F_i \leftarrow$ fetchFunctions($W_i$)
3:     $\lambda(p), \ell(p) \leftarrow$ getInfo($W_i$)
4:     for $f_k \in F_i$ do
5:         $f_t \leftarrow time() + f_i^d + coldStart$
6:         if $f_t > uT$ then
7:         end if
8:         $fork$(TAROT − PLACEMENT($\mathcal{C}, k+1, W_i, uT, \lambda(p), \ell(p)$))
9:         $deploy(f_k, \Pi_{i,k})$
10:    end for
11: end function
```

B. Placement Service

Finding Π_i requires to identify, for each function $f_i \in F_i$, a node $c \in \mathcal{C}$ that allows deployment of f_i in terms of resources' availability and minimization of T, which requires to iterate over the whole set of compute nodes $\mathcal{C}$. T depends on $\mathcal{U}$ and $\mathcal{U}^c$ depend on the user subscriptions, which depend on user locations [11]. Therefore, predicting future UD's locations allows to predict future users' subscriptions and consequently future $\mathcal{U}$ and $\mathcal{U}^c$, which affect future latency T^f.

Placement service is described by Algorithm 2. After initialization of T^f, we extract the sorted functions in W using `fetchFunctions`. In lines 7-14, TAROT-PREDICT (Algorithm 3) is invoked to identify the node minimizing T for function f, T^f, until time $time() + \mathtt{uT}$. Computation of the new placement starts from the i-th function, where i is given as input during the workflow execution. Goal of placement service is to ensure that $\Pi_{i,k}$ minimizes constraints violation during the time window. To this end, placement service relies on the prediction service, TAROT-PREDICT, to predict future average latency for single function f, T^f. First, algorithm checks if node c has enough resources to execute function f_k (line 8). Then, if the allocation on node c reduces T_f, T_f value is updated (line 10) and placement of f_k is updated in line 11. Finally, Π_i is updated for all functions in W.

Algorithm 2 PLACEMENT algorithm.

```
1: function TAROT − PLACEMENT($\mathcal{C}$, $func$, $W$, uT, $\lambda(p)$, $\ell(p)$)
2:     $T^f \leftarrow \infty$
3:     $F \leftarrow$ fetchFunctions(W)
4:     $j \leftarrow 0$
5:     for $j < |W|$ do
6:         $k \leftarrow func$
7:         for $c \in \mathcal{C}$ do
8:             if isCompatible($f_k, c$) then
9:                 if TAROT − PREDICT($c, f_k, time() +$ uT, $\lambda(p), \ell(p)$) $< T^f$ then
10:                    $T^f \leftarrow$ TAROT − PREDICT($c, f_k, time() +$ uT, $\lambda(p), \ell(p)$)
11:                    $\Pi_{i,k} \leftarrow c$
12:                end if
13:            end if
14:        end for
15:        $j \leftarrow j + 1$
16:        $k \leftarrow k + 1\ MOD\ |W|$
17:    end for
18: end function
```

C. Prediction Service

Typical prediction approaches [24], [25] are based on AI and require training of complex models, requiring a huge amount of data, which might be unsuitable for Edge nodes due to their limited storage and computational capabilities [26]. As a consequence, we design a prediction service based on Kalman filters. Kalman filter is a data-driven prediction approach which estimates the state of a dynamic system starting from a set of measurements collected from noisy sensors. In our case, location data are collected through each UD's GPS coordinates. The state of each vehicle n at time j is represented as a vector $\texttt{coords}(n, j) = \langle x_n^j, y_n^j, v_{x,n}^j, v_{y,n}^j \rangle^T$, where x_n^j and y_n^j represent respectively the latitude and longitude of UD n at time j and $v_{x,n}^j$, $v_{y,n}^j$ the velocity on x and y axis. Kalman filter is composed of two phases: the *predict*, where $\texttt{coords}(n, j)$ is estimated based on previous state $\texttt{coords}(n, j-1)$. To predict the $\texttt{coords}(n, j)$, we need to consider motion equation (including acceleration) and the correlation between previous position. State transition is modeled by state transition matrix $F(n, j)$, $F(n, j)$ matrix is defined as $\begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$. Where Δt models the measurement interval. Since velocity of each UD depends on external influence (i.e., UD accelerates/decelerates), we have to take it into account to improve accuracy of predictions. To this end, we use the control matrix $B(n, j)$ and the acceleration vector $\vec{a}_j$, which is defined as $\begin{pmatrix} \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} \\ 1 \\ 1 \end{pmatrix} \vec{a}_j$. We assume that acceleration $\vec{a}_j$ follows a normal distribution with mean μ and standard deviation σ for each coordinate. Correlation between $\texttt{coords}(n, j)$ and $\texttt{coords}(n, j-1)$ is defined by the covariance matrix W, which defines how variations in each component correlation between each component of $\texttt{coords}(n, j)$. Since we assume that variation on x have no influence on y, considering motion equation we define $W(n, j)$ as $\begin{pmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{pmatrix} \sigma$.
Finally, we want to use real measurements to update our model. Measurements are defined by the observation matrix $H(n, j)$ and random measurements noise $V(j)$. We assume that $V(j)$ is constant and equal to 10 meters, as typical in GPS sensors. Measurements are updated according to the Kalman gain $K(n, j)$, which is calculated according to Equation 18.

$$K(n, j) = W(n, j-1) H^T(n, j) \cdot \\ \cdot (H(n, j) W(n, j-1) H^T(n, j) + V(j)). \quad (18)$$

State update equation for each UD n is then defined as

$$\texttt{coords}(n, j) = \texttt{coords}(n, j-1) + \\ + K(\vec{z}(n, j) - H(n, j)\texttt{coords}(n, j-1)), \quad (19)$$

where $\vec{z}(n, j)$ is the real measurements of n location.

Prediction of T^f is described in Algorithm 3. Its input are target node c, the function f_i to be deployed on c and the temporal horizon of prediction, uT. First, topics to which function subscribes/publishes are collected in lines 2-5. In the for loop from line 8-12, Equation 19 is used to predict future location of publisher/subscribers $n \in S' \cup P'$, starting from the location at time() until target time. Finally, future average latency for f_i on node c, T^f is returned.

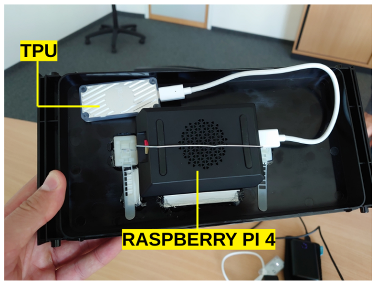
Algorithm 3 Prediction of T^f.

```
1: function TAROT − PREDICT($c, f_i$, target, $\lambda(p), \ell(p)$)
2:     $t_{in} \leftarrow topic^{in}(f_i)$
3:     $t_{out} \leftarrow topic^{out}(f_i)$
4:     $S' \leftarrow \Theta(\mathcal{S}, t_{in})$
5:     $P' \leftarrow \Theta(\mathcal{P}, t_{out})$
6:     $\vec{z} \leftarrow$ getMeasurements($\mathcal{S}$)
7:     for $n \in S' \cup P'$ do
8:         for $j \in$ [time(), target do
9:             coords$(n)^j \leftarrow$ Equation 19
10:            $K(n, j) \leftarrow$ Equation 18
11:        end for
12:    end for
13:    $T^f \leftarrow$ Equation 17
14: return $T^f$
15: end function
```

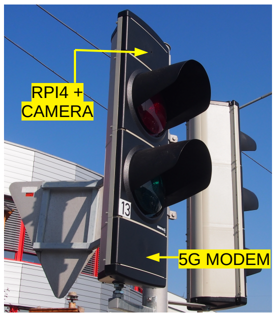
V. Experimental Setup

A. Data Collection

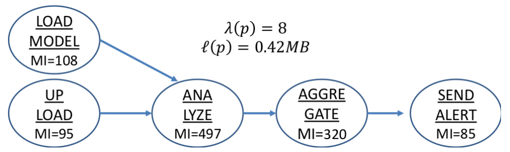
To design an accurate simulation, we use data collected on InTraSafEd5G infrastructure, deployed on two traffic lights at a crossroad in Vienna. Each node is composed of an RPI V2 camera and a Coral USB TPU, attached to a Raspberry Pi 4B+ as shown in Figure 6a. To execute serverless functions, we installed OpenFaaS 0.7.3 and Python 3.7.3. Edge nodes are attached to traffic lights as shown in Figure 6b. Data are used to simulate a large scale deployment of TAROT on different areas of Vienna, as it is common in evaluation of large scale edge systems [27]. The structure of workflows is illustrated in Figures 7a and 7b together with $\lambda(p)$, $\ell(p)$, and $\texttt{MI}(f)$.



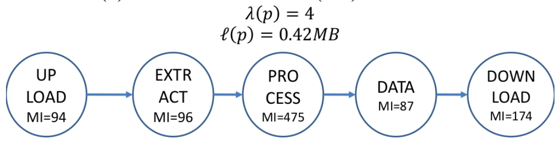
(a) InTraSafEd5G Edge Node.



(b) Node Deployment.



(a) `InTraSafEd5G (IS)` Workflow



(b) `MobiAR (AR)` Workflow

Fig. 7: Target Serverless Workflows.

1) Workflows: The InTraSafEd5G workflow processes video frames coming from cameras installed at different crossroads, to detect pedestrians in drivers' blind spots. First, cameras upload video frames under specific topics (`UPLOAD`), which are then analyzed by an object detection function (`ANALYZE`) once the detection model has been loaded (`LOAD_MODEL`). Results are then aggregated (`AGGREGATE`) and alerts are sent to all UD registered to the topic. During this workflow execution, UD subscribe to all topics collecting information about the crossroad closest to the UD.

`MobiAR` workflow describes a touristic augmented reality application which shows information about points of interest (POI) on a UD (i.e., phone, smart glasses). First, image of the POI is uploaded from UD to the infrastructure (`UPLOAD` function), then image metadata are extracted by the `EXTRACT` function. Metadata are then processed (`PROCESS` function) and data are obtained by an external service (`DATA`). Finally, data are delivered to the UD (`DOWNLOAD`). During this workflow execution, UD publish the picture under a topic and then subscribe to the topic related to POI data.

$\lambda(p)$ depends on application: for example, each camera on average forwards 4 key frames out of 25 per second to save network bandwidth, therefore $\lambda(p) = 4$ for *AR*. For InTraSafEd5G, $\lambda(p) = 8$ because workflows aggregates data from at least two cameras. The value of $\ell(p)$ is based on the average size of frames collected by each camera. $\texttt{MI}(f)$ value depends on function execution time, that is obtained through code instrumentation and averaged over 100 executions.

2) Network Measurements: We collect latency and bandwidth measurements on the InTraSafEd5G infrastructure for TAROT simulation. Since network communication in InTraSafEd5G relies on the MQTT protocol, as is common in IoT [28], we collect measurements of MQTT latency through a self-designed android application. For bandwidth measurements, we use `iperf`. We simulate Edge and Cloud deploying MQTT broker and iperf server in different locations, as in [10]. We collect data on 3G, 4G and 5G network.

B. Simulation Setup

After evaluating different simulators, i.e., iFogSim [29] and EdgeCloudSim [30], we decided to use SLEIPNIR [31], which provides validated models for cloud/edge infrastructure.

1) Compute Nodes: The $\texttt{MIPS}(c)$ value of cloud and edge nodes as well as other hardware capabilities are selected to reflect our experiment testbed. Table II shows hardware specifications. For cloud nodes, we use the same number and

Node type	Number	CPU	`MIPS`
Cloud	6	64	2500
Edge	$\{36, 100, 144\}$	4	2000
Publishers	$\{42, 135, 172\}$	✗	✗
Subscribers	$\{360, 1000, 1440\}$	✗	✗

TABLE II: Configuration of Compute Nodes.

locations of [31], which provides a realistic estimation for this scenario. Edge nodes are placed on the map for each traffic light, as in [14].

2) Network Infrastructure: Due to the unreliability of connections in mobile data distribution services [32], we model connections between nodes using random variables whose distribution is based on data collected in InTraSafEd5G, as described in Section V-A2. UD are connected to the infrastructure via 3G, 4G, or 5G. We employ $Q(t)$ variable, whose distribution is summarized in Table III, to determine which connection is available at time t. Latency and bandwidth used for a single-hop are summarized in Table III. Bandwidth corresponds to $r(n_i, n_j)$ in Table I.

Connection	Edge QoS		Cloud QoS		$Q(t)$
	Latency [ms]	Bandwidth [Mbps]	Latency [ms]	Bandwidth [Mbps]	
3G	371.16	24.1	378.7	10.5	0.28
	519	1.05	561	1.05	0.04
4G	54.2	52.1	109.8	10.5	0.48
	106	48.16	163	21	0.1
5G	45.475	56.2	86.09	48.32	0.09
	74	22	105.5	22	0.01

TABLE III: 1-hop Network QoS Distribution.

3) Mobility Simulation: We use mobility traces generated by the SUMO simulator [33] for subscribers' movements. SUMO is a state-of-the-art mobility simulator offering simulation of vehicular and pedestrian traffic on OpenStreetMap maps. We extract maps of three different areas of Vienna: `HERNALS`(H) (11.35 km^2), `LEOPOLDSTADT`(L) (19.27 km^2), and `SIMMERING`(S) (23.23 km^2). Then, SUMO is used to simulate traffic on the selected areas similar to publications on Vehicular Ad-Hoc Networks (VANET), e.g., [3].

We collect logs about relative coordinates of each vehicle moving in the aforementioned areas with a sampling interval of 1 second. UD movement is simulated by updating their coordinates according to the logs as simulation advances.

VI. Results

We evaluate TAROT by simulating placement on the areas described in Section V-B3. Each result is averaged over 1000 iterations. We run experiments on TU Vienna cluster using Spark v3.0.1 and Java 1.8. Each node features 48 Intel(R) Xeon(R)E5-2650 v4 2.20GHz cores.

A. Parameter Study

TAROT predictions are influenced by the value of uT, which determines the temporal horizon over which predictions of T are performed. In this set of experiments, we evaluate runtime and T with respect to uT. Comparison is performed with a ORACLE approach, where instead of TAROT predictions (described in Algorithm 3) the real SUMO traces are used.

B. Comparative Evaluation

We compare TAROT to other state-of-the-art placement heuristics, FFDPROD [34] and COSTLESS [35], the state-of-the-art list-scheduling heuristic PEFT [36], to show that improvements in T are not related to workflow scheduling, and the ORACLE approach previously described. We compare placement solutions obtained by these algorithms on selected workflows (Figure 7). We vary $\ell(p)$ between 1kb and 1Mb, which represent realistic estimations of sizes of expected data objects. Results are shown, respectively, in Figures 8a for InTraSafEd5G and in Figures 8c for AR. For comparative analysis, when increasing $\ell(p)$, TAROT provides a 46% lower T in comparison to FFDPROD and 20.6% less in comparison to the COSTLESS algorithm for InTraSafEd5G (Figure 8a) as well as up to 25% less for AR (Figures, 8c).

C. Discussion

Results of Section VI-A, described by Figures 8a-8d show that TAROT is capable to compute a target node in less than 150ms for both workflows, which allows to satisfy latency constraints of target applications. The difference in execution time between ORACLE and TAROT is related to the fact that while ORACLE just performs a lookup in the mobility traces, while TAROT performs calculation of Equations 19, 18 for a longer time. Concerning uT, selecting a $uT = 120$s allows to achieve a solution close to ORACLE (average 8.4% difference), which is the average time needed by each UD to move in the range of another Edge node.

Concerning the comparative analysis (Section VI-B), we can see that TAROT is capable to reduce T up to 46% with respect to $\ell(p)$, (Figures 9b-10c). Also, we notice that results of TAROT are comparable with the ORACLE approach, which employs the real values from the traces instead of predicted value. These simulation results show the benefits of TAROT approach for T and its applicability to IoT data processing. Moreover, the shortest runtime in comparison to centralized solutions (i.e., PEFT and COSTLESS) provides a promising results towards real-world TAROT implementation.

VII. Related work

Smart city applications and their requirements are described in [26], [37].

Serverless computing is described in [38]. In [18], different applications of serverless computing in Cloud are described. In [17], several issues related to the implementation of serverless computing on Fog infrastructures are discussed, while exploitation of serverless features for IoT Cloud applications has been proposed in [4]. In [28], [39], data distribution for IoT systems is addressed only for Cloud resources. Benefits of serverless paradigm for hybrid Edge/Cloud infrastructures has been discussed in [5], while [6] described development of tinyFaaS, a serverless framework for Edge/Cloud infrastructures. The use of edge computing to improve performance of IoT processing has been proposed, e.g., by [40]. Still in the context of FaaS architectures, [41] focuses in provisioning and allocation of functions in Edge/Cloud architectures.

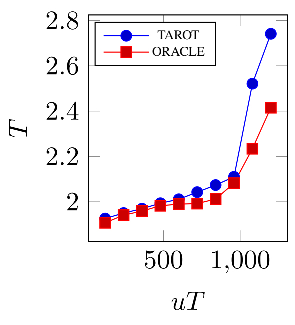
The use of IoT messaging protocols in the fog/edge domain is described in [42]. A pub/sub scheme for IoT data distribution in the context of Software Defined Networks is presented in [43]. Other works have addressed data distribution for latency-constrained applications, e.g., [44] in terms of load balancing and [45] in terms of cost-effectiveness. MQTT-based Middleware for edge-based IoT applications has been proposed by [11], [46]. In addition to data delivery, we target data processing on cloud/edge infrastructure.

The papers [19] describe FBase, a function replication service for data-intensive fog applications, without considering latency requirements and UDs' mobility. In [35], COSTLESS placement approach for serverless functions is described. Placement of FaaS workflows is discussed in [47], focusing mostly on data protection. An auction-based function placement approach to optimize latency and cost is proposed by [7], without considering FaaS workflows and mobility. Dynamic placement of functions for mobile serverless applications is described by [8], but focuses more on security issues.

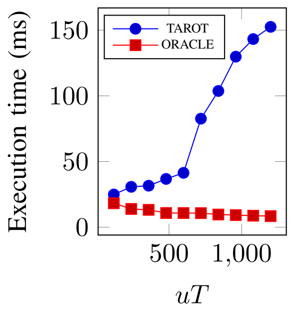
VIII. Conclusion and Future Work

In this paper, we proposed the TAROT approach for latency-aware function placement in serverless edge computing systems. We evaluated TAROT in a simulated urban deployment and compared it with other state-of-the-art algorithms. Results show that TAROT provides latency improvements up to 46% compared to alternatives.

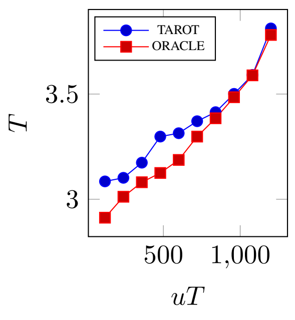
In future work, we plan to develop TAROT approach in combination with hybrid smart contracts, to address privacy issues typical of IoT scenario. Moreover, we plan to evaluate its performance on existing serverless computing frameworks, i.e., tinyFaaS and OpenWhisk. Finally, we will investigate data privacy provided by TAROT, which is of particular interest due to the use of video footages and UD's GPS.



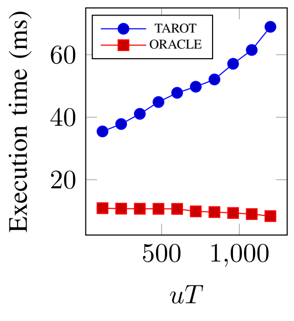
(a) W=IS, T w.r.t. uT.



(b) W=IS, RT w.r.t. uT.

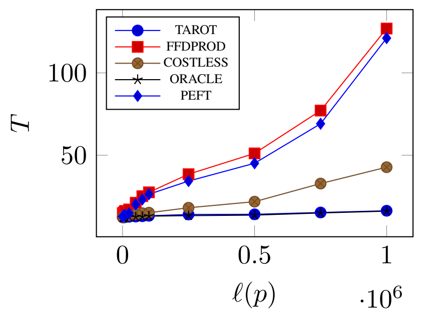


(c) W=AR, T w.r.t. uT.

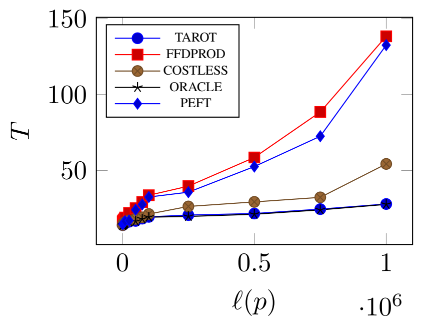


(d) W=AR, RT w.r.t. uT.

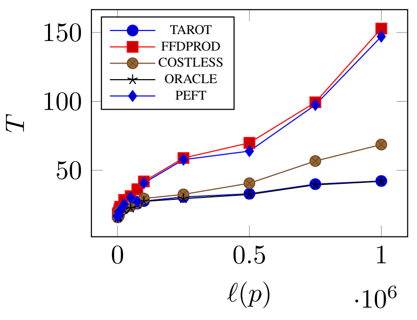
Fig. 8: Parameter Studio, $W = \{\mathtt{IR}, \mathtt{AR}\}$, $uT = [120, 1200]$s, $M = H$.



(a) T w.r.t. $\ell(p)$, $M = H$.

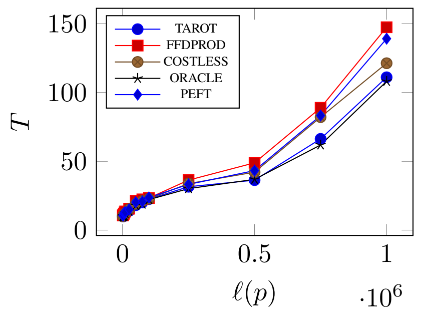


(b) T w.r.t. $\ell(p)$, $M = L$

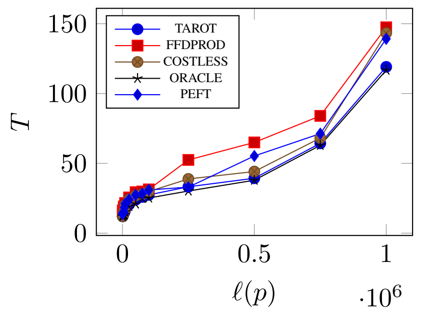


(c) T w.r.t. $\ell(p)$, $M = S$.

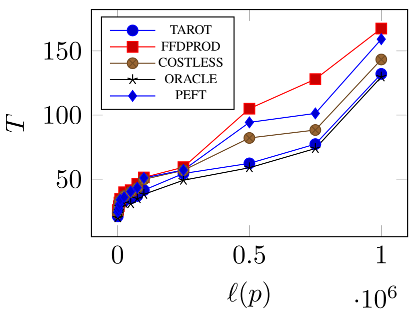
Fig. 9: Comparative study, $W = \{\mathtt{IS}\}$, $\ell(p) = [1e^3 - 1e^6]$ bytes, M=$\{H, L, S\}$.



(a) T w.r.t. $\ell(p)$, M=H.



(b) T w.r.t. $\ell(p)$, M=L.



(c) T w.r.t. $\ell(p)$, M=S.

Fig. 10: Comparative study, $W = \{\mathtt{AR}\}$, $\ell(p) = [1e^3 - 1e^6]$ bytes, M=$\{H, L, S\}$.

Acknowledgements

This work has been partially funded through the Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015, the CHIST-ERA grant CHIST-ERA-19-CES-005, by the Austrian Science Fund (FWF): I 5201-N, and 5G Use Case Challenge InTraSafEd 5G (Increasing Traffic Safety with Edge and 5G) funded by the City of Vienna.

References

[1] F. Pallas, P. Raschke, and D. Bermbach, "Fog computing as privacy enabler," in *Internet Computing*. IEEE, 2020.

[2] E. B. Smida, S. G. Fantar, and H. Youssef, "Video streaming challenges over vehicular ad-hoc networks in smart cities," in *2017 International Conference on Smart, Monitored and Controlled Cities (SM2C)*, Feb 2017, pp. 12–16.

[3] V. D. Maio, R. B. Uriarte, and I. Brandic, "Energy and profit-aware proof-of-stake offloading in blockchain-based vanets," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2019, Auckland, New Zealand, December 2-5, 2019*, K. Johnson, J. Spillner, D. Klusácek, and A. Anjum, Eds. ACM, 2019, pp. 177–186.

[4] T. Pfandzelter and D. Bermbach, "Iot data processing in the fog: Functions, streams, or batch processing?" in *Proc. of DaMove*. IEEE, 2019.

[5] L. Baresi and D. F. Mendonça, "Towards a serverless platform for edge computing," in *Proc. of ICFC*. IEEE, 2019.

[6] T. Pfandzelter and D. Bermbach, "tinyFaaS: A lightweight faas platform for edge environments," in *Proceedings of the Second IEEE International Conference on Fog Computing (ICFC 2020)*. IEEE, 2020.

[7] D. Bermbach, S. Maghsudi, J. Hasenburg, and T. Pfandzelter, "Towards auction-based function placement in serverless fog platforms," in *Proceedings of the Second IEEE International Conference on Fog*

Computing (ICFC 2020). IEEE, 2020.

[8] A. Bocci, S. Forti, G.-L. Ferrari, and A. Brogi, "Secure faas orchestration in the fog: how far are we?" *Computing*, vol. 103, no. 5, pp. 1025–1056, 2021.

[9] D. Pinto, J. P. Dias, and H. Sereno Ferreira, "Dynamic allocation of serverless functions in iot environments," in *2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*, 2018, pp. 1–8.

[10] I. Lujic, V. D. Maio, K. Pollhammer, I. Bodrozic, J. Lasic, and I. Brandic, "Increasing traffic safety with real-time edge analytics and 5g," in *EdgeSys@EuroSys 2021: 4th International Workshop on Edge Systems, Analytics and Networking, Online Event, United Kingdom, April 26, 2021*, A. Y. Ding and R. Mortier, Eds. ACM, 2021, pp. 19–24.

[11] J. Hasenburg and D. Bermbach, "GeoBroker: Leveraging geo-contexts for IoT data distribution," *Computer Communications*, 2020.

[12] A.-S. Karakaya, J. Hasenburg, and D. Bermbach, "SimRa: Using crowdsourcing to identify near miss hotspots in bicycle traffic," *Elsevier Pervasive and Mobile Computing*, 2020.

[13] D. Marimon, C. Sarasua, P. Carrasco, R. Álvarez, J. Montesa, T. Adamek, I. Romero, M. Ortega, and P. Gascó, "Mobiar: Tourist experiences through mobile augmented reality," *Information and communication technologies in tourism*, 01 2010.

[14] A. Aral, V. De Maio, and I. Brandic, "Ares: Reliable and sustainable edge provisioning for wireless sensor networks," *IEEE Transactions on Sustainable Computing*, pp. 1–1, 2021.

[15] G. Luo, Q. Yuan, H. Zhou, N. Cheng, Z. Liu, F. Yang, and X. S. Shen, "Cooperative vehicular content distribution in edge computing assisted 5g-vanet," *China Communications*, vol. 15, no. 7, pp. 1–17, 2018.

[16] M. Elhoseny, "Intelligent firefly-based algorithm with levy distribution (ff-l) for multicast routing in vehicular communications," *Expert Systems with Applications*, vol. 140, p. 112889, 2020.

[17] B. Cheng, J. Fürst, G. Solmaz, and T. Sanada, "Fog function: Serverless fog computing for data intensive iot services," *CoRR*, vol. abs/1907.08278, 2019.

[18] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless Computing: One Step Forward, Two Steps Back," in *Proceedings of CIDR*, Jan. 2019.

[19] J. Hasenburg, M. Grambow, and D. Bermbach, "Towards A Replication Service for Data-Intensive Fog Applications," in *Proceedings of the 35th ACM Symposium on Applied Computing, Posters Track (SAC 2020)*. ACM, 2020.

[20] J. Hasenburg, F. Stanek, F. Tschorsch, and D. Bermbach, "Managing latency and excess data dissemination in fog-based publish/subscribe systems," in *Proceedings of the Second IEEE International Conference on Fog Computing (ICFC 2020)*. IEEE, 2020.

[21] M. A. Prada, P. Reguera, S. Alonso, A. Morán, J. J. Fuertes, and M. Domínguez, "Communication with resource-constrained devices through mqtt for control education," *IFAC-PapersOnLine*, vol. 49, no. 6, pp. 150 – 155, 2016.

[22] Q. Fan and N. Ansari, "Towards workload balancing in fog computing empowered iot," *IEEE Transactions on Network Science and Engineering*, 2018.

[23] T. Han and N. Ansari, "A traffic load balancing framework for software-defined radio access networks powered by hybrid energy sources," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 1038–1051, 2016.

[24] Z. S. Nejad, H. Heravi, A. R. Jounghani, A. Shahrezaie, and A. Ebrahimi, "Vehicle trajectory prediction in top-view image sequences based on deep learning method," *arXiv preprint arXiv:2102.01749*, 2021.

[25] Z. Zhao, H. Fang, Z. Jin, and Q. Qiu, "Gisnet:graph-based information sharing network for vehicle trajectory prediction," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7.

[26] I. Lujic, V. D. Maio, and I. Brandic, "Efficient edge storage management based on near real-time forecasts," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, May 2017, pp. 21–30.

[27] J. Xie, C. Qian, D. Guo, X. Li, S. Shi, and H. Chen, "Efficient data placement and retrieval services in edge computing," in *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*, 2019, pp. 1029–1039.

[28] K. Beckmann and O. Dedi, "sdds: A portable data distribution service implementation for wsn and iot platforms," in *2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES)*, 2015, pp. 115–120.

[29] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments," *CoRR*, vol. abs/1606.02007, 2016.

[30] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 39–44.

[31] V. De Maio and I. Brandic, "Multi-objective mobile edge provisioning in small cell clouds," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 127–138.

[32] A. Aral and I. Brandic, "Dependency mining for service resilience at the edge," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 228–242.

[33] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. WieBner, "Microscopic traffic simulation using sumo," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 2575–2582.

[34] X. Li, A. Ventresque, J. Murphy, and J. Thorburn, "A fair comparison of vm placement heuristics and a more effective solution," in *2014 IEEE 13th International Symposium on Parallel and Distributed Computing*, June 2014, pp. 35–42.

[35] T. Elgamal, A. Sandur, K. Nahrstedt, and G. Agha, "Costless: Optimizing cost of serverless computing through function fusion and placement," *CoRR*, vol. abs/1811.09721, 2018.

[36] H. Arabnejad and J. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 682–694, 03 2014.

[37] R. J. Wang, X. Li, S. Ao, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," *CoRR*, vol. abs/1804.06882, 2018.

[38] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2017, pp. 405–410.

[39] Y. Ma, J. Rao, W. Hu, X. Meng, X. Han, Y. Zhang, Y. Chai, and C. Liu, "An efficient index for massive iot data in cloud environment," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, ser. CIKM '12. New York, NY, USA: ACM, 2012, pp. 2129–2133.

[40] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. A. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving iot from the cloud." in *HotStorage*. USENIX, 2015.

[41] O. Ascigil, A. Tasiopoulos, T. K. Phan, V. Sourlas, I. Psaras, and G. Pavlou, "Resource provisioning and allocation in function-as-a-service edge-clouds," *IEEE Transactions on Services Computing*, pp. 1–1, 2021.

[42] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, p. 116, 2019.

[43] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif, "Publish/subscribe-enabled software defined networking for efficient and scalable iot communications," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 48–54, 2015.

[44] J. Gascon-Samson, F. Garcia, B. Kemme, and J. Kienzle, "Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud," in *2015 IEEE 35th International Conference on Distributed Computing Systems*, 2015, pp. 486–496.

[45] V. Setty, R. Vitenberg, G. Kreitz, G. Urdaneta, and M. v. Steen, "Cost-effective resource allocation for deploying pub/sub on cloud," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, 2014, pp. 555–566.

[46] T. Rausch, S. Nastic, and S. Dustdar, "Emma: Distributed qos-aware mqtt middleware for edge computing applications," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 191–197.

[47] S. Kotni, A. Nayak, V. Ganapathy, and A. Basu, "Faastlane: Accelerating function-as-a-service workflows," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Jul. 2021, pp. 805–820. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/kotni