ORCH: Distributed Orchestration Framework using Mobile Edge Devices

Klervie Toczé
Dept. of Computer and Information Science
Linköping University, Sweden
klervie.tocze@liu.se

Simin Nadjm-Tehrani
Dept. of Computer and Information Science
Linköping University, Sweden
simin.nadjm-tehrani@liu.se

***Abstract*—In the emerging edge computing architecture, several types of devices have computational resources available. In order to make efficient use of those resources, deciding on which device a task should execute is of great importance.**

Existing works on task placement in edge computing focus on a resource supply side consisting of stationary devices only. In this paper, we consider the addition of mobile edge devices. We explore how mobile and stationary edge devices can augment the original task placement problem with a second placement problem: the placement of the mobile edge devices.

We propose the ORCH framework in order to solve the joint problem in a distributed manner and evaluate it in the context of a spatially-changing load. Our implementation of the combined task and edge placement algorithms shows a normalized 83% delay-sensitive task completion rate compared to a perfect edge placement strategy.

***Index Terms*—Fog/edge computing, resource management, edge mobility, task placement, edge placement**

I. Introduction

The emergence of scenarios with Internet-of-everything in smart cities, smart transportation, and smart manufacturing creates use cases where short response times and control over data in motion are cornerstones [1]. Edge or fog computing is promoted as a paradigm to address the timeliness, energy efficiency and security requirements in such contexts. Although many labels for this emergent trend exist, including those related to 5G networks, what they share is the notion of bringing the computational resources closer to the user [2].

Deploying an edge infrastructure means placing numerous edge devices over large areas. This is done for example in vehicular networks by creating road side units, or by co-locating edge devices with radio base stations for other scenarios [3]. Each of those edge devices will have a certain range where it can offer its services and multiple edge devices can be grouped together to collaborate, e.g. in fog colonies [4].

Once resources are present at the edge, the next step is to match the application requests with some of the available resources. This matching corresponds to solving a *load placement* problem, i.e. where to place the task at hand for computation. Recently, this problem has been studied in the context of sensors and actuators by Skarlat et al. [4], or for generic mobile end users by Wang et al. [5].

This work was supported by the Swedish national graduate school in computer science (CUGS).

In most of the current works, the edge devices supplying the resources considered are stationary [2]. This resource type has many advantages but it may not be very flexible and easy to deploy in case of sudden changes in the locality of the arriving load. At the other end of the mobility spectrum, serving the load surges with *mobile* edge devices is very flexible regarding locality but it requires careful design of distributed algorithms to give good coverage without wasting resources.

Serving unpredicted load surges in distributed locality over time cannot be done by a fine spreading of stationary or mobile edge devices due to cost, space, and other (urban) restrictions. Therefore, a combination of mobile and stationary edge devices is an option to explore as a middle ground.

Some edge applications will have a time-space varying task generation with spikes, e.g. augmented or virtual reality, which also have high real-time constraints [6]. Thus, the load should be processed very close to the end device to avoid too high communication overhead, high service latency, and degradation of quality of service (QoS).

When enlarging the supply side with mobile edge devices, a placement problem for the mobile edge devices is added to the load placement problem. We refer to this as the Distributed Dynamic Task and Edge Placement (D2TEP) problem.

In this paper, we present a distributed orchestration framework that enables the edge devices to provide a high QoS under temporally and spatially changing load. More precisely, the contributions of our work are:

- A distributed edge orchestration framework called ORCH tackling the D2TEP problem for the first time. The framework is populated with a first attempt to provide instances of deadline-aware algorithms and functions, but it is a generic framework to be used for further studies.
- An extension of EdgeCloudSim [7] implementing the ORCH framework and associated schemes.
- An evaluation of the implemented ORCH algorithms in comparison with three alternative strategies, where QoS measured as number of completed tasks before deadline, and edge utilization are used as evaluation metrics.

In order to show that higher QoS in the presence of local load surges is achievable with some mobile edge devices that can move on-demand, we evaluate ORCH in comparison with: 1) a systems with only stationary edge devices, 2) a system with a pre-scheduled edge movements strategy, and 3) a system with a perfect edge movement strategy.

The rest of the paper is organized as follows. Section II describes the D2TEP problem and the different models and assumptions. Next, Section III presents the ORCH framework and associated functions and algorithms. Section IV introduces the ORCH implementation. Section V presents experimental results. Finally, we discuss related works in Section VI and conclude the paper in Section VII.

II. Problem Description

This section presents the D2TEP problem followed by the motivational scenario and the different models adopted.

A. Distributed Dynamic Task and Mobile Edge Placement Problem

The D2TEP problem is a two-fold problem: a task placement subproblem and a mobile edge placement subproblem.

The **task placement subproblem** is the following: for each task r incoming to the closest edge device e, e has to decide on which edge device e' the task r will be allocated so that e' has enough resources to compute r before its deadline.

The **mobile edge placement subproblem** is the following: at every load change, a location for all the mobile edge devices should be found such that the incoming load is served with as high success ratio as possible.

The D2TEP problem is distributed since task placement can be done by all the edge devices, and it is dynamic as the mobile edge placement is dynamically updated depending on the changes in the load volume and locality over time.

B. Motivational scenario

This work is placed in the context of a smart city where edge infrastructure providers have to handle load coming from mobile end devices. The edge infrastructure providers control edge devices that are used to serve this load.

The load consists of tasks with various requirements coming from end devices with mobility over various locations in the city space. The load depends on what the end user is currently doing and where she is located. Tasks are described in more detail in Section II-F but the challenge in this scenario is that some tasks have very strict real-time requirements. Not satisfying those requirements amounts to unacceptable QoS. This means that those tasks have to be serviced in the vicinity of the end devices issuing them in order to keep the round trip latency as low as possible to accommodate the short deadlines.

C. System Model

In this work, we use a three-layer system model based on the coordinator device architecture proposed in an earlier survey [2]. Our model is presented in Figure 1. There are three types of devices involved in this model, one type per layer:

- Cloud device (stationary, dark blue cloud)
- Edge device (mobile or stationary, yellow rectangles)
- End device (mobile or stationary, orange triangles)

A dashed outline in Figure 1 indicates a mobile device.

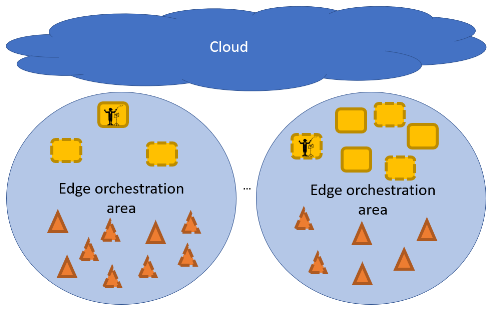


Fig. 1: System model considered for the ORCH framework.

In addition, we introduce the notion of *edge orchestration area* to represent the spatial distribution of end devices connecting to the system and to reflect the way the generated load will reach the edge devices in that space. The cloud services are common for the edge orchestration areas but each end or edge device only belongs to one edge orchestration area. This work focuses on one edge orchestration area, but it can trivially be scaled up to several areas. It may also be extended to overlapping areas with some notion of handover, but we begin with the simplest instance.

Each edge orchestration area is composed of M end devices and a set of E edge devices $\mathcal{E} = \{e_1, ..., e_E\}$. A subset of $\mathcal{E}$ denoted as $\mathcal{E}^{mob}$ contains the mobile edge devices. At any point in time, one edge device $e \in \mathcal{E}$ acts as the edge device responsible for the area orchestration in each area. This edge device is referred to as *area orchestrator* in the ORCH framework. It is depicted as a rectangle with a conductor icon in Figure 1 and will be denoted o in the rest of the paper. The cloud device for the edge orchestration area is denoted c.

Within an edge orchestration area, each edge device e considers a subset of $\mathcal{E}$ as edge devices that can be used for offloading a part of the load coming to e. Those edge devices are called *neighboring devices*.

D. Area model

This model considers that edge devices are only able to serve requests when located at specific positions called *serving positions*, and not while moving between those. For example, the edge device could be a drone that has two modes: either its resources (e.g. energy and computation) are used for serving edge requests at a serving position, or its resources are used for navigating between two of those serving positions.

Moreover, in the context of a smart city, the exact location of an end device is not required as long as it is in the connectivity range of some edge device. Therefore the geographical area is represented by S *segments*, where $\mathcal{S} = \{s_1, ..., s_S\}$ and where the physical area included within segment s_i corresponds to the connectivity range of an edge device e located at the serving position ρ_i within s_i. The set of adjacent segments to segment i is denoted as $\mathcal{A}_i$.

Figure 2 shows an example of how a real neighborhood at Vallastaden in Linköping could be divided according to this model. The serving positions are denoted as orange circles and the segments are delimited in black. In a real environment,

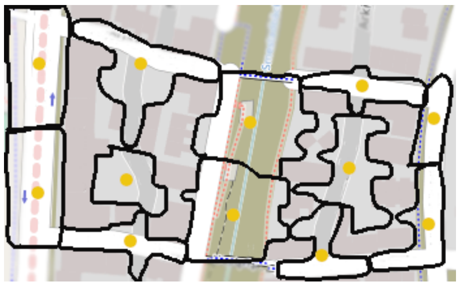


Fig. 2: Example of division of a neighborhood into segments.

the shape of segments will depend on the hardware used for communication and how the physical environment looks like. Details on how this is done are out of the scope of this paper, where we assume that the segments are already defined.

Each edge device e is associated with the segments that it can serve. Those segments are denoted as the set $\mathcal{S}_e \subseteq \mathcal{S}$. A mobile edge device can only serve the segment that it is currently located in (hence for $e \in \mathcal{E}^{mob}$, $|\mathcal{S}_e| = 1$), whereas stationary edge devices can serve several segments. It is assumed that 1) all end and mobile edge devices know their current segment, and 2) an end device m located in segment s^m has a mechanism to establish communication with the closest edge device. The *closest edge device* is defined as a mobile edge device located within the same segment, and if none, a stationary edge device serving this segment.

E. Resource Models

The first resource type in this model is computational resources. Those are considered at the edge and cloud level. Cloud level resources are assumed unlimited but edge resources are limited, for example, to a fixed number of Millions of Instructions Per Second (MIPS). The duration of the computation at a device thus depends on the size of the task to execute and the resources available at the device.

The second resource type is communication resources shared among all the devices. In this work, we assume unconstrained bandwidth in the system, meaning that communication time is equal to the transmission time between two devices. The model can later be extended by adding bandwidth constraints and queuing time.

The transmission time or delay d_L for a task to move between devices depends on the type of the link L and is defined as follows for the different links considered. Equation 1 shows the delay for a link L that connects an end device m in segment s to an edge device e.

$$d_L = \begin{cases} d_{short} & \text{if } s^m \in \mathcal{S}_e \text{ and } |\mathcal{S}_e| = 1 \\ d_{medium} & \text{if } s^m \in \mathcal{S}_e \text{ and } |\mathcal{S}_e| > 1 \\ \infty & \text{otherwise} \end{cases} \tag{1}$$

The infinite delay models the case where an end device is outside of the coverage areas for the edge devices.

The delay between two edge devices is assumed to be similar to what it takes for sending a task from an end device to a stationary edge device, i.e. d_{medium}.

Finally, the delay for a link between an edge device e and the cloud device c is as follows:

$$d_L = \begin{cases} d_{medium} + d_{long} & \text{if } |\mathcal{S}_e| = 1 \\ d_{long} & \text{otherwise} \end{cases} \tag{2}$$

In an orchestration area, d_{short}, d_{medium}, and d_{long} are configurable parameters.

Edge devices can communicate to exchange relevant information (corresponding to signaling). This is assumed to have a negligible impact on the task communication times.

F. Load Model

The load model includes a task model and an application model. For the task model, this work considers that each task r is independent and is characterized by (a) a computation requirement R_r in Millions of Instructions (MI) and (b) a deadline D_r in milliseconds. The characterization of tasks can be extended to include other resource requirements. The generation of tasks is modeled as a Poisson process.

In this model, the tasks fall into three categories, depending on their deadline requirements. Those categories are derived from the traffic classes presented by Bianzino et al. [8]:

- delay-sensitive (DS) tasks
- delay-constrained (DC) tasks
- delay-tolerant (DT) tasks

Tasks falling into the DS category need to be executed on an edge device located in the same segment as the end device in order to avoid communication link delays. In our study, this means it needs to execute on a mobile edge device or on a stationary edge device dedicated to this segment. An example of applications that could produce DS tasks are augmented or virtual reality applications.

Tasks falling into the DC category need to be executed on some edge device, but not necessarily an edge device in this segment. Examples of edge applications that could generate DC tasks are edge analytics applications such as face recognition in video-based surveillance.

Tasks falling into the DT category can be executed on an edge or cloud device. An example of applications producing DT tasks are delay-tolerant gaming applications.

For the application model, the above task categories are used to model two application types used by end devices: urgent applications and regular ones. Urgent applications are modeled by a generated task mix with a large proportion of DS and DC tasks, and few DT tasks, whereas regular applications are modeled by a task mix with a larger portion of DC tasks, few DT and no DS tasks. Moreover, urgent applications have a lower mean for the Poisson process generating the tasks.

End device mobility and applications launched by users naturally change the locality of the load over time. In this work, we refer to a sudden surge in DS traffic at a given segment as a *load change*.

G. End Device Mobility Model

To model the mobility of end devices, we adapt the Weighted Waypoint model presented by Hsu et al. [9] to the area model presented in Section II-D. Indeed, end devices are only able to move to neighboring segments so their movements between segments cannot be considered fully random as in the widely used Random Waypoint model.

Each segment has its own stay duration distribution which represents how long the end devices will stay in that segment, depending on the characteristics of the segment. For example, a mobile end device will probably stay less time in a segment with only apartment buildings than in a segment with shops. On entering a segment, an end device gets a value for stay duration from the segment's distribution. This work considers three types of segments: residential, commercial, and outside segments but the list can be modified or extended in different scenarios. The stay duration in each segment is characterized as an exponential distribution with two different rates: a small one λ^s for residential segments and a large one λ^l for commercial and outside segments.

In this model, end devices can only move to adjacent segments and the transition probability P_{ij} from segment i to an adjacent segment $j \in \mathcal{A}_i$ is the same towards each adjacent segment, meaning it is calculated as the inverse of $|\mathcal{A}_i|$. For end devices, the transition between two neighboring segments is instantaneous as the time for the end device to move close to the segment border is modeled as part of the stay duration.

H. Edge Device Mobility Model

The mobility model for edge devices is a two-state model. Mobile edge devices can be *stable*, when they are located at a serving position ρ and accept tasks. They can also be *moving* between a service position ρ and a service position ρ', not necessarily a neighboring one. In this state, they are unavailable for the system, and are therefore not considered when determining the closest edge device or performing mobile edge placement since they do not accept new tasks while changing segments.

The current model does not support migration of tasks, meaning that once a task coming from an end device m has been placed on an edge device e for execution, all the execution will be performed at e. When the result is available, if e and m are not located in the same segment anymore because one of them moved, the task is considered as failed. Extending the model would need to borrow the concept of store-and-forward from delay-tolerant networking.

III. ORCH FRAMEWORK

The orchestration framework ORCH, shown in Figure 3, is proposed to address the D2TEP problem. We begin with an overview and then describe each component in more detail.

A. Overview

The ORCH framework comprises two different orchestrations: the edge orchestration and the area orchestration.

Each edge device performs the *edge orchestration*, which addresses the task placement subproblem of D2TEP and is depicted in Figure 3 with blue boxes. The *task characterizer* identifies characteristics of the incoming tasks, and the

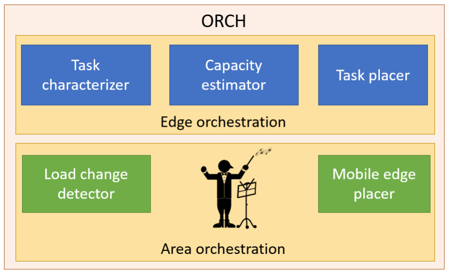


Fig. 3: Overview of the ORCH framework.

capacity estimator estimates the available capacity of the neighboring edge devices. Both are used by the *task placer*, which decides on which edge device the task is going to be executed. Based on the received tasks, each edge device also stores load information to be used by the area orchestrator. The edge orchestration takes place each time a task is submitted to an edge device (or a configurable batch of tasks is to be considered at once).

One edge device per orchestration area is designated as area orchestrator at relevant decision points. In addition to the edge orchestration, this edge device also performs the *area orchestration* (depicted in Figure 3 by green boxes) in order to address the mobile edge placement subproblem of the D2TEP. The area orchestration components are run asynchronously when load changes require it. The area orchestrator role can change according to various schemes, e.g. using a rotation among the edge devices, with higher priority given to the stationary ones (or the ones with more available resources). This rotation scheme will not be discussed further in the current paper but can be similar to existing schemes [8]. The two components of the area orchestrator are the *load change detector*, which observes the area load and detects load changes upon which the *mobile edge placer* decides where the mobile edge devices should be moved to. After the mobile edge placement is performed, the edge devices move to their new locations (when needed).

In this work, a strict deadline-aware approach is used to address the D2TEP problem. This means that the task requirements described in Section II-F are strictly enforced and deadline misses lead to task abortion. For example, short deadlines for DS tasks means that an edge device will be dedicated to serving in the segment where a surge is detected. If the deadline requirements cannot be accommodated, because there is no available resource to fulfill the needs, the task is deemed failed.

The ORCH framework is designed and implemented in a modular way so that the detailed algorithms and functions used in each component can be modified easily (see Section IV). It is thus possible to address the D2TEP problem using refinements or variations to the schemes proposed below.

B. Edge Orchestration

The edge orchestration proposal is presented in Algorithm 1. It has two main parts: the first one (lines 1-6) corresponds to the task characterizer and task placer components. A count of DS tasks received per segment is maintained. Other information, such as task placement success ratio, could be stored if needed by the area orchestrator. The second part (lines 7-10) corresponds to the capacity estimator component.

Algorithm 1: Edge orchestration

```
   // Runs at each edge device $e$
   parameter: $T_c$ // Cap. est. period
   state     : $C = << n, c >>$ // List of cap.
               est. $c$ for each neighboring $n$
               $L = << s, \#DS >>$ // List of # DS
               tasks for each segment $s$
1  upon receiving task $r$ from segment $s$ do
       // Task characterizer
2      $tag \leftarrow$ Characterize($r$);
3      if $tag$ = $DS$ then
4          Increment $\#DS$ for $s$
       // Task placer
5      $device \leftarrow$ PlaceTask($r$, $tag$, $e$);
6      sendForExecution($r$, $device$)
7  every $T_c$ do // Capacity estimator
8      $\mathcal{N} \leftarrow$ GetNeighboringEdgeDevices($e$);
9      foreach $n \in \mathcal{N}$ do
10         Update $c$ for $n$ with GetAvailRes($n$);
```

1) Task Characterizer: This will associate a deadline category to the tasks as a means of preparing them before being considered for placement. For the DS category, this is done by considering whether $D_r < 2 * d_{medium}$, in which case r is tagged as DS. A similar reasoning with d_{long} is used to tag a task as DC. The remaining tasks are tagged as DT.

2) Capacity Estimator: Computational resources are aggregated in a pool and measured as units of available VM. The capacity estimator will periodically (with period T_c) fetch the percentage of the computational resource pool currently not in use from all the neighboring edge devices (lines 7-10, Algorithm 1). The current periodic fetching can be easily modified to have it event-triggered instead.

3) Task Placer: A deadline-aware closest-first task placement strategy presented in Function 2 is used. The general idea of the function is to place the tasks spatially as close as possible to the requesting end device in order to avoid unnecessary communication overhead. Hence, the function will try placing the task 1) on the receiving edge device, which is the closest edge device and the entry point for any end device request, 2) on a neighboring edge device if the closest device (in space) does not have enough resources according to the capacity estimator and 3) on the cloud otherwise. In this paper, the considered neighboring devices are an edge device able to serve the whole area if the receiving device is an edge device serving only one segment, but this could be extended, for example to allow for more placement attempts.

The only exception is when the task is of category DT and the receiving/neighboring edge device is a mobile edge device (line 3/line 9). As mobile edge resources are scarce and are

Function 2: Deadline-aware closest-first task placement

```
   parameter: $\tau_a$// Avail. capacity threshold
   input    : $r, tag, e$ // Task, tag, rec. edge
   output   : $device$ // Placement for $r$
1  Function PlaceTask($r, tag, e$)
       // Try to place $r$ on receiving $e$
2      if GetAvailCapacity($e$) $\geq \tau_a$ then
3          if Not($tag$ = DT and $e$ is mobile) then
4              Place $r$ on $e$;
5      else
           // Try to place $r$ on neighbor edge
6          $\mathcal{N} \leftarrow$ GetNeighboringEdgeDevices($e$);
7          foreach $n \in \mathcal{N}$ do
8              if GetAvailCapacity($n$) $\geq \tau_a$ then
9                  if Not($tag$ = DT and $n$ is mobile) then
10                     Place $r$ on $n$;
11     if No placement found at the edge and $tag$ = DT
         then Place $r$ on the cloud;
12     else Failure to place $r$;
```

the only ones capable of handling DS tasks, they should be preferably used for those task types. DT tasks are therefore only placed on stationary edge devices or cloud devices.

Once a placement decision has been taken, the task is transmitted to the executing device, which may or may not be the same as the receiving device. When getting an execution request, the edge device puts it directly on its execution list.

C. Area Orchestration

The area orchestration proposal is provided in Algorithm 3. The load change detector runs periodically (with detection period T_d) but this could be modified to have event-triggered change detection. Each area orchestrator also keeps track of the current high load segments as a set $\mathcal{H}$ of ordered segments and of the current placement of the mobile edge devices as a list P of $< e, s >$ for each $e \in \mathcal{E}^{mob}$.

Algorithm 3: Area orchestration

```
   parameter: $T_d$ // Detection period
   state    : $\mathcal{H}$ // High load segments
              $P$ // Mobile edge placement
   // Runs at each area orchestrator $o$
1  every $T_d$ do
       // Load change detector
2      <$isChanging$,$\mathcal{H}$>$\leftarrow$LdChangeDetected($o$,$\mathcal{H}$);
3      if $isChanging$ then
           // Mobile edge placer
4          $newPlace \leftarrow$ PlaceMobileEdge($o$, $\mathcal{H}$, $P$);
5          Instruct devices to move when needed;
6          Update $P$ with $newPlace$;
```

1) Load Change Detector: The load change detector is described in Procedure 4 and has four parts: a load aggregation part, a prediction part, a high load identification part, and a change detection part.

The load aggregation part (lines 4-6) fetches the load information stored as part of the edge orchestration at every edge device during the previous detection period to determine the load per segment in the whole area.

Procedure 4: Load change detection including prediction

```
parameter: $\tau_h$// High load threshold
input    : $o$ // Area orchestrator
           $curH$ // Current high load segs.
output   : $isChanging$ // Detection result
           $hlSegs$ // New high load segs.
state    : $areaL = << s, \#DS >>$ // # DS
           tasks per segment $s$ in the area
1  Procedure LdChangeDetected($o$, $curH$)
2      $\mathcal{S} \leftarrow$ GetSegmentsInArea($o$);
3      $\mathcal{E} \leftarrow$ GetEdgeDevicesInArea($o$);
       // Aggregate the DS load per segment
4      foreach $e$ in $\mathcal{E}$ do
5          foreach $s$ in $\mathcal{S}$ do
6              Increment $\#DS$ for $s$ in $areaL$ by
                 GetLoadInfo($e$, $s$);
7      foreach $s$ in $\mathcal{S}$ do
           // Predict DS load
8          $pred \leftarrow$ FUSD_Prediction($s$, $\#DS$ for $s$
             from $areaL$);
           // Identify high load segments
9          if $pred \geq \tau_h$ then
10             Insert $s$ to $hlSegs$ so that segments are
                 ordered with highest load first ;
       // Detect load change
11     if $hlSegs \neq curH$ then
12         $isChanging \leftarrow$ true;
```

For the prediction part (line 8), an adaptation of the Fast Up and Slow Down (FUSD) algorithm proposed by Xiao et al. [10] is used. Instead of predicting the percentage of CPU utilization, the FUSD algorithm is used to predict the estimated number of DS tasks in each segment in the next detection period. The predictor keeps load and prediction information for a certain number of previous periods, called the window W. In order for the prediction to capture the increase and decrease of the resource need in a satisfactory way, one parameter is used when the load is increasing ($\uparrow \alpha$) and another when the load is decreasing ($\downarrow \alpha$).

For identification (lines 9-10), a similar approach to the hot spot classification by Xiao et al. is used. Segments with high predicted load are identified using a threshold τ_h and sorted with highest load first. Finally, a load change is detected (lines 11-12) if the set of ordered high load segments has changed since the previous detection period.

It is part of future work to use adaptive ways of performing the load change detection, e.g. with machine learning.

2) Mobile Edge Placer: The second area orchestration component focuses on serving as many DS tasks as possible. It considers the ordered set of segments with high DS load that was computed in the load change detector. The general idea is to associate the segment with highest load with the mobile edge device currently located in the segment with the lowest DS load, until no more mobile edge devices are available. This is performed in three steps presented in Function 5.

Function 5: Deadline-aware mobile edge placement

```
input : $o$ // Area orchestrator
        $curH$ // Current high load segs.
        $curP$ // Curr. mob. edge placement
output: $newP$ // New mobile edge placement
1  Function PlaceMobileEdge($o$, $curH$, $curP$)
2      $availME \leftarrow$ GetSortedAvailMobileEdge($o$);
       // Keep existing placements as much
          as possible
3      foreach $s$ in the first $|availME|$ high load segments
         do
4          if $s$ is already associated to $e$ in $curP$ then
5              Add $< e, s >$ to $newP$ ;
6              Remove $e$ from $availME$ and $s$ from $curH$;
       // Associate remaining edge devices
          with remaining high load segments
7      foreach $s \in curH$ do
8          if there is a remaining mobile edge device then
9              Add $<$ Head($availME$), $s >$ to $newP$ ;
10             Remove Head($availME$) from $availME$;
```

First (line 2), the available mobile edge devices, i.e. those not in the moving state, ordered by lowest DS load at the current segment are fetched. The number of currently available mobile edge devices (denoted as $|availME|$) will constrain how many high load segments the ORCH can serve.

Then (lines 3-6), the mobile edge placer determines if some of the first $|availME|$ high load segments (those with highest load) are already associated with an edge device in the current placement. Those associations are kept to avoid unnecessary unavailable periods due to mobile edge movements.

Finally (lines 7-10), the remaining mobile edge devices, if any, are associated with the remaining high load segments.

IV. ORCH IMPLEMENTATION

In order to evaluate the ORCH framework, we use and extend EdgeCloudSim [7], an edge environment simulation tool built upon CloudSim [11]. The tool was adapted to the models used in this work by implementing the mobility of edge devices, the area model with segments, and the ORCH framework structure with corresponding algorithms and functions, among other minor adaptations.

The Java code for ORCH is available online[1]. The extension includes 6 new abstract classes (one for the ORCH orchestrator

[1]https://gitlab.liu.se/ida-rtslab/public-code/2019_orch

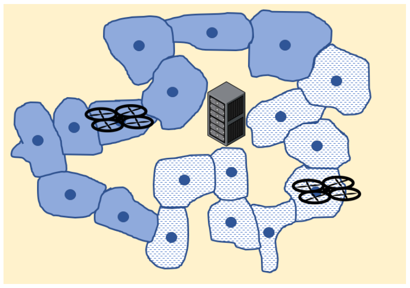


Fig. 4: Schematic view of the considered neighborhood.

and five for the ORCH framework components) that allow for modularity, and the corresponding 6 classes corresponding to the implementation of the proposal. Moreover, 3 additional classes were implemented corresponding to the models for the load generation, the end device mobility, and the edge device mobility. Two additional setup files were created (for the area model and the application model) as new variations of the original EdgeCloudSim simulator. The different parameters used in the experiments are added to the parameter file for easy modifications when conducting experiments.

Building a distributed mobile edge extension to EdgeCloudSim was facilitated by its modular design. However, implementing and integrating the new models and the new scenarios was a challenge that required a deep understanding of both EdgeCloudSim and CloudSim, due to complex interactions. Moreover, implementating the ORCH framework in a modular way to ease testing new variants of the components or experimenting with alternative models added to the implementation complexity.

According to our experiments, scaling up and down the size of the experiment area and of the edge infrastructure is relatively straightforward and does not have a strong impact on the simulator running time, but the total number of tasks generated by the end devices does. The maximum number of tasks submitted in the experiments in Section V was 155458 and the running time for this run was 1 min 50 sec. On average, the simulator running time is less than 1 min 30 sec when simulating 20 min of experiment.

V. Evaluation

In this section, we show the performance of the ORCH framework with the proposed algorithms and functions in terms of QoS and edge utilization in presence of different loads. We compare this to three baseline strategies.

A. Performance metrics

The task and edge placement outcome is evaluated using a metric that reflects success or failure of a task completing in time denoted as *completed tasks* below. More specifically:

- success ratio per task category
- number of completed tasks per device type

Looking at the outcome per task category is important given that a high QoS requires that the success ratio for DS tasks should be as high as possible.

The outcome per device type shows on which device type the completed tasks were executed. The number of tasks completed on edge devices should be high (especially on mobile ones) and the number of tasks completed in the cloud should be low. This is natural as it was the very motivation for using edge devices and mobile ones in particular, to create added QoS compared to a classic cloud.

B. Baseline strategies

1) For task placement: We compare the ORCH task placement strategy with a *First Fit* strategy. In this strategy, the task is placed on the receiving edge device if resources suffice, otherwise on the neighbouring edge device if enough resources and otherwise on the cloud. First Fit strategies are commonly used in recent research as a baseline for task placement strategies [4], [12], [13].

2) For edge mobility: We compare the ORCH edge mobility strategy with three baselines: 1) a strategy where all edge devices are stationary, called *Stationary*, 2) a perfect edge mobility strategy, called *Perfect*, and 3) a simpler edge mobility strategy, called *Scheduled*.

For all four strategies the total amount of resources (i.e. CPU available) is the same. The resources are equally distributed among stationary edge devices serving the whole area (called *area-wide*) and edge devices serving one segment each (called *segment-wide*). For the Stationary strategy, the segment-wide edge devices are stationary, whereas they are mobile for the Scheduled, Perfect and ORCH strategies.

The Stationary strategy corresponds to the current research in edge/fog resource management, where the mobility of edge devices is not considered. In the Perfect strategy, it is assumed that some mobile edge device exactly follows the movements of an end device running urgent applications, as long as there are mobile edge devices available.

In the Scheduled strategy, the mobile edge devices move from one segment to another in a pre-defined pattern (e.g. as an edge device in a scheduled bus might do, with bus stops acting as serving positions). In the following experiments, they move clockwise around the orchestration area, starting from a random segment. Büchel and Corman [14] showed that the best fitting distribution for stop duration at bus stops and traveling time between bus stops is a lognormal distribution, based on empirical data. Therefore, the duration of stable and moving states (both in seconds) for Scheduled is randomly selected from two lognormal distributions.

C. Experimental setup

A neighborhood composed of one outside segment and one edge orchestration area with eighteen segments is considered. The edge resources consist of one area-wide device and two segment-wide devices.The experimental setup can be changed to increase the scale of the edge infrastructure (at the cost of increased simulation time) but the chosen scale is similar to state-of-the-art service placement research [4], [13].

The neighborhood is illustrated in Figure 4. The edge orchestration area is composed of segments being represented

TABLE I: Simulation parameters.

Area	Parameter	Value
Generic	Simulated time	1200000 ms (20 min)
	# end devices (M)	25
Computation	Edge device	500 MIPS
	Cloud device	12000 MIPS
	CPU share per task	20%
Communication delays	d_{short}	1 ms
	d_{medium}	25 ms
	d_{long}	100 ms
Tasks	Computation req. (R_r)	10000 MI
	DS task deadline (D_r)	30 ms
	DC task deadline (D_r)	150 ms
	DT task deadline (D_r)	300 ms
Urgent applications	Task mix (DS/DC/DT)	40%/50%/10%
	Poisson mean	50 ms
Regular applications	Task mix (DS/DC/DT)	0%/70%/30%
	Poisson mean	500 ms
Segments	Rate λ^s	36000 ms
	Rate λ^l	300000 ms
Edge mobility	Scheduled: stable state duration distribution	Lognormal($\mu = 2.5$, $\sigma^2 = 0.49$)
	Scheduled: moving state duration distr.	Lognormal($\mu = 3.3$, $\sigma^2 = 0.04$)
	ORCH: moving state duration distribution	Gamma($k = 7.5$, $\theta = 4$)
Capacity estimator	Period T_c	1000 ms
Task placer	Threshold τ_a	5
Load change detector	Period T_d	1000 ms
	Window size W	8
	FUSD percentile	90th
	$\uparrow \alpha$	-0,2
	$\downarrow \alpha$	0,7
	Threshold τ_h	5

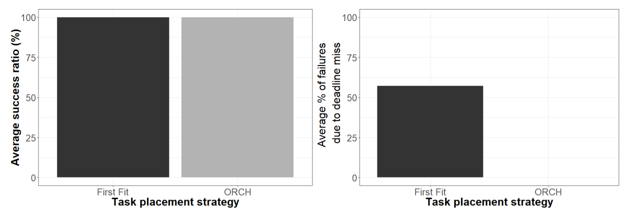
as blue shapes. Half of the segments are residential segments (light dashed shapes) and the rest of the segments are commercial segments (dark colored shapes). The serving positions are denoted with dark blue circles, and the outside segment is represented in light yellow.

For each run of all four strategies, the movement of each end device is randomly generated and some end devices are randomly selected for running urgent applications. All end devices create tasks following the load model of Section II-F if they are located in a residential or commercial segment. This creates different surge locations and duration for every run.

Table I summarizes all the parameters and corresponding values considered in the simulation. The simulation uses milliseconds as a unit of time, contrary to previous work using seconds [4], [7]. This reflects the edge context better. In particular, communication delays in 5G are in the order of a few ms [15], which is reflected in d_{short}. Changing the values related to the edge context may impact the results. However, the aim of this study was not to optimize the round trip time in a specific scenario, but rather to show that a higher success ratio is achieved when using the ORCH mobile edge concept.

D. Load

When evaluating the system, we consider two load scenarios. In the first one, the number of end devices running urgent applications is the same as the number of available segment-wide edge devices (i.e. 2). Hence the load is *balanced* with regards to resources but the load surge varies in space over time. In the second scenario the number of end devices running urgent applications (set to 6) exceeds the number of available segment-wide edge devices, hence creating a system level *overload.*



(a) on success ratios for DS tasks (b) on failures due to deadline miss

Fig. 5: Impact of task placement strategy (balanced load).

TABLE II: Avg (with st. dev.) and min/max # of successful DS tasks for the balanced load (B) and overload (O) scenarios.

Strategy	Average		σ		Min		Max	
	B	O	B	O	B	O	B	O
Stationary	2097	4681	2418	4189	0	0	8831	15366
Scheduled	751	2362	381	600	99	1345	1747	3851
ORCH	10103	19854	3356	4639	2934	9416	16195	28305
Perfect	11881	21816	3336	5967	4097	12580	17099	31660

E. Impact of the task placement strategy

To isolate the impact of the task placement strategy, we fix the edge placement strategy to Perfect for the following experiment. We perform 50 runs per task placement strategy.

Figure 5a shows the average success ratios for DS tasks. It is similar for both task placement strategies (ORCH and First Fit) and close to 100% as the Perfect edge mobility strategy ensures that the mobile edge devices are following the devices running urgent applications. Therefore, it appears that ORCH task placement does not add to QoS for DS tasks. However, when looking at the average percentage of failed tasks that are due to a deadline miss (shown on Figure 5b), it is 57% for the First Fit and 0% for ORCH. This means that the ORCH strategy is able to avoid wasting resources by executing tasks on devices when those tasks are deemed to miss their deadline. This is possible because the task placer in ORCH knows the type of the tasks thanks to the task characterizer and can avoid placements that will for sure lead to a deadline miss.

The results for the overload scenario show similar behaviors and are omitted due to space limitations.

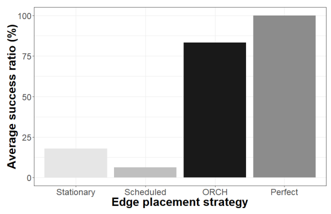
F. Impact of the edge placement strategy

We perform 50 runs per edge placement strategy, using the ORCH task placement strategy in all edge devices.

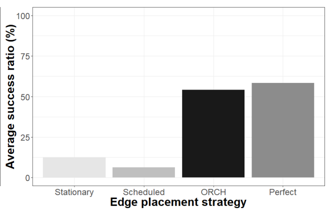
1) Balanced load scenario: Figure 6a shows the success ratios of DS tasks for the balanced load study with all the baselines. Key indicators for this study are shown in Table II.

Regarding the DS tasks, ORCH outperforms Scheduled and Stationary with a 13 times, respectively 4 times higher average success ratio for DS tasks (83.3% vs 6.3%, respectively 17.8%). ORCH also comes as close as 83% to the Perfect performance which succeeds for 99.9% of the DS tasks.

Regarding the DC and DT tasks, the results (not shown due to space limitation) are similar for the four strategies, with a



(a) Balanced load



(b) Overload

Fig. 6: Success ratios for DS tasks.

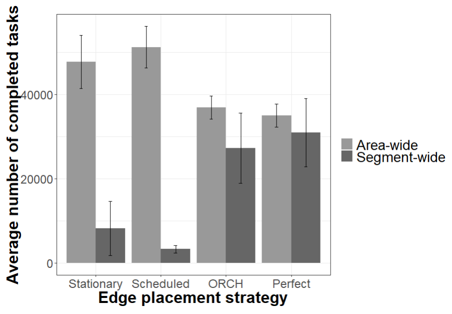


Fig. 7: Division of completed tasks among mobile/stationary edge devices (balanced load scenario).

success ratio very close or equal to 100%. This is expected as all strategies handle those tasks in a similar way.

Figure 7 shows the second metric of evaluation: average number of tasks completed on each edge device type in the balanced scenario. The number of tasks completed in the cloud is negligible for all four strategies (the resources of edge devices are enough to handle the load) and is therefore not shown. The strategies can be divided into two groups. First, Stationary and Scheduled have a minority (15%, respectively 6%) of tasks completed in the segment-wide devices. On the opposite, Perfect and ORCH exhibit a balanced division with 43%, respectively 47% of tasks completed in the segment-wide devices. A balanced division is good because the segment-wide devices have a higher utilization (less waste of resource) and the area-wide ones have more free resources for handling a higher load without having to resort to cloud resources.

2) Overload scenario: Figure 6b shows the success ratios of DS tasks for the different edge placement strategies with key indicators summarized in Table II.

In this highly overloaded scenario where there are potentially three times more demand than supply, the success ratio for DS tasks is now on average 12.6% for the Stationary strategy, 6.3% for the Scheduled strategy, 54.1% for the ORCH strategy, and 58.4% for the Perfect strategy. The results indicate that while Scheduled and Stationary continue to be incapable to manage the load, ORCH handles 92% of the Perfect service rate in this overly loaded scenario. Moreover, ORCH handles 8 times more DS tasks than Scheduled on average and always at least 144% more tasks than the highest Scheduled performance. The average success ratios for DC and DT tasks are similar to the previous ones, with a slight decrease for the DC tasks (at most 2%, not shown here).

With regard to the division of completed tasks among device types, the results for the overload scenario were comparable to those for the balanced load one and are therefore not presented.

G. Discussion and possible extensions

Regarding task placement, a next step is to include task migration as part of the task placer role, so that the edge device can modify the current task placement dynamically when receiving a new task, in case this would improve the QoS or the edge utilization.

Regarding edge placement, DS tasks are mostly rejected when there is no available segment-wide edge device. This either happens because the segment-wide edge devices are moving between serving positions (Scheduled/ORCH), or they are not at the surge location, either because of their pre-defined schedule/fixed location (Scheduled/Stationary) or because the load change has not yet been detected (ORCH). The reported results where ORCH outperforms Scheduled and Stationary show the importance of having mobile segment-wide devices as well as intelligent placing strategies, so that they are available as much as possible and where they are needed.

Moreover, the results vary (see the standard deviation in Table II) depending on how the load for the actual run looks like, i.e. whether end devices generating DS traffic are in the same segment or are spread all over the orchestration area, and whether those end devices stay in the orchestration area during the whole experiment or are subject to churn. Further improvement of the results is therefore possible through adoption of adaptive algorithms.

VI. Related Works

Rana et al. [16] found that increased latency has a stronger impact on application execution time when placing services on both edge and cloud than on only cloud, hence supporting the importance of moving edge devices very close to the end devices, as studied in this work.

Solving task placement problems in edge computing was one of the most active areas in a recent survey [2]. In the majority of the works covered by that survey, a central entity gets all the task placement requests. In this type of scenario, the task placement problem can be formulated as an optimization problem where several tasks have to be dispatched among several edge devices and is solved using various optimization techniques [4], [5], [17], [18]. Some works proposed heuristics instead of optimization [12], [13], but the task placement is still performed in a centralized manner. Foggy [19] is one example framework for centralized orchestration. In our work, the task placement is fully distributed, i.e. each edge device is responsible for placing the tasks it receives.

Distributed task placement problems are typically tackled by a pool of edge devices when the needed resources not present at the current edge device. Guiguis et al. [20] present a distributed way of maintaining the resource information, but contrary to our work where we place one task at a time, they optimize the placement of a set of tasks. Mascitti et al. [21] find the best composition of services between different devices, whereas in our work, a task will be executed by only one device.

Singh et al. [22] and Fizza et al. [23] also propose task placement algorithms using tags to categorize the different tasks. However, contrary to our work, the tags are related to security/privacy characteristics and not deadlines.

The deadline-aware approach that we used, where DT tasks are prevented to run on mobile edge devices to keep the resources available for DS or DC tasks is conceptually similar to the delay-priority policy of Bittencourt et al. [24]. In addition to being used in another context, our approach considers tasks arriving one at a time and thus they cannot be prioritized upon reception.

Regarding the edge placement subproblem, to the best of our knowledge, it has only been tackled for placing stationary edge device (e.g. in [3]), but not for mobile ones.

VII. Conclusion

Using a mix of mobile and stationary edge devices provides the ability to serve all edge task categories in presence of local sudden surges in load in a flexible manner, i.e. without having to change the infrastructure in place. This is especially interesting when the delay-sensitive traffic is present.

Our analysis of the ORCH orchestration shows promises. We see that its components, while relatively straightforward when taken separately, have the power to achieve high quality of service and edge utilization when combined. The actual implementation of the framework in a modular manner with free code distribution provides the potential for studying more elaborate models by the research community in the future. Our current work includes implementing an augmented reality application that utilizes an edge node, in order to validate the timeliness requirements assumed for simulations.

Among directions for future works, we consider the inclusion of adaptive algorithms, as well as characterizing the overheads of signaling and of various strategies by elaborating cost models. An obvious trade-off emerging from such models would be against the consumed energy. In addition, while simulations are relevant as a means of understanding scenarios with many actors and serving components, a further proof-of-concept in a physical testbed would be an idea for future extension of the work.

References

[1] A. Bartolí, F. Hernández, L. Val, J. Gorchs, X. Masip-Bruin, E. Marín-Tordera, J. Garcia, A. Juan, and A. Jukan, "Benefits of a coordinated fog-to-cloud resources management strategy on a smart city scenario," in *Euro-Par 2017: Parallel Processing Workshops*, 2018, pp. 283–291.

[2] K. Toczé and S. Nadjm-Tehrani, "A taxonomy for management and optimization of multiple resources in edge computing," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 7 476 201:1–7 476 201:23, 2018.

[3] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *IEEE International Conference on Edge Computing (EDGE)*, 2018, pp. 66–73.

[4] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized IoT service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, 2017.

[5] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, 2017.

[6] Z. Tan, Y. Li, Q. Li, Z. Zhang, Z. Li, and S. Lu, "Supporting mobile VR in LTE networks: How close are we?" *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 1, pp. 8:1–8:31, 2018.

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

[8] A. P. Bianzino, M. Asplund, E. J. Vergara, and S. Nadjm-Tehrani, "Cooperative proxies: Optimally trading energy and quality of service in mobile devices," *Computer Networks*, vol. 75, pp. 297–312, 2014.

[9] W.-J. Hsu, K. Merchant, H.-W. Shu, C.-H. Hsu, and A. Helmy, "Weighted waypoint mobility model and its impact on ad hoc networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 1, pp. 59–63, 2005.

[10] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107–1117, 2013.

[11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.

[12] V. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sànchez-López, J. Garcia, G. Ren, A. Jukan, and A. J. Ferrer, "Towards a proper service placement in combined fog-to-cloud (F2C) architectures," *Future Generation Computer Systems*, vol. 87, pp. 1 – 15, 2018.

[13] Y. Xia, X. Etchevers, L. Letondeur, A. Lebre, T. Coupaye, and F. Desprez, "Combining heuristics to optimize and scale the placement of iot applications in the fog," in *IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, 2018, pp. 153–163.

[14] B. Büchel and F. Corman, "Modelling probability distributions of public transport travel time components," in *18th Swiss Transport Research Conference (STRC)*, 2018.

[15] S. Shankland. (2018, Dec) How 5G aims to end network latency. [Online]. Available: https://www.cnet.com/news/how-5g-aims-to-end-network-latency-response-time/

[16] O. Rana, M. Shaikh, M. Ali, A. Anjum, and L. Bittencourt, "Vertical workflows: Service orchestration across cloud & edge resources," in *IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2018, pp. 355–362.

[17] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (QoE)-aware placement of applications in fog computing environments," *Journal of Parallel and Distributed Computing*, 2018.

[18] T. Ouyang, Z. Zhou, and X. Chen, "Follow Me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.

[19] D. Santoro, D. Zozin, D. Pizzolli, F. D. Pellegrini, and S. Cretti, "Foggy: A platform for workload orchestration in a fog computing environment," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 231–234.

[20] M. Guiguis, Q. Gu, T. Penner, L. Tammineni, T. Langford, A. Rivera-Longoria, A. Johnson, and B. V. Slyke, "Assignment and collaborative execution of tasks on transient clouds," *Annals of Telecommunications*, vol. 73, no. 3, pp. 251–261, 2018.

[21] D. Mascitti, M. Conti, A. Passarella, L. Ricci, and S. K. Das, "Service provisioning in mobile environments through opportunistic computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 12, pp. 2898–2911, 2018.

[22] A. Singh, N. Auluck, O. Rana, A. Jones, and S. Nepal, "RT-SANE: Real time security aware scheduling on the network edge," in *10th International Conference on Utility and Cloud Computing (UCC)*, 2017, pp. 131–140.

[23] K. Fizza, N. Auluck, O. Rana, and L. Bittencourt, "PASHE: Privacy aware scheduling in a heterogeneous fog environment," in *IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2018, pp. 333–340.

[24] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.