Cost-optimized, data-protection-aware offloading between an edge data center and the cloud

Zoltán Ádám Mann, Andreas Metzger, Johannes Prade, Robert Seidl, and Klaus Pohl

Abstract—An edge data center can host applications that require low-latency access to nearby end devices. If the resource requirements of the applications exceed the capacity of the edge data center, some non-latency-critical application components may be offloaded to the cloud. Such offloading may incur financial costs both for the use of cloud resources and for data transfer between the edge data center and the cloud. Moreover, such offloading may violate data protection requirements if components process sensitive data. The operator of the edge data center has to decide which components to keep in the edge data center and which ones to offload to the cloud, with the objective of minimizing financial costs, subject to constraints on latency, data protection, and capacity. In this paper, we formalize this problem and prove that it is strongly NP-hard. To address this problem, we introduce an optimization algorithm that (i) is fast enough to be run online for dynamic and automatic offloading decisions, (ii) guarantees that the solution satisfies hard constraints on latency, data protection, and capacity, and (iii) achieves near-optimal costs. We also show how the algorithm can be extended to handle multiple edge data centers. Experiments performed with up to 450 components show that the cost of the solution found by our algorithm is on average only 2.7% higher than the optimum. At the same time, our algorithm is very fast: it optimizes the placement of 450 components in less than 300 milliseconds on a commodity computer.

Index Terms—edge computing, fog computing, edge data center, offloading, resource optimization, data protection

1 Introduction

Many new applications need to process large volumes of data from distributed end devices, e.g., sensors [1], [2]. Processing these data solely in the end devices is often not feasible because of the devices' limited compute and storage capacity. Offloading the data processing to cloud data centers solves the capacity problem, but leads to other concerns, an important one being communication latency.

Edge computing (aka. fog computing) provides cloud-like services with low latency [3], [4]. In edge computing, small-scale *edge data centers* deployed in close proximity to the end devices offer higher capacity than the end devices. Edge data centers can host application components that process data from nearby end devices [5], [6]. On the other hand, non-latency-critical components may be offloaded to the cloud instead of the edge data center, to benefit from the virtually unlimited capacity of the cloud [7], [8].

Problem. We focus on an edge data center, hosting a set of applications [9]. Each application consists of components (e.g., microservices). The edge data center offers virtualized resources for hosting the components, e.g., in containers. Although the capacity of the edge data center is typically larger than the capacity of end devices, it is still limited [10], [11]. If the load of the edge data center exceeds its capacity, some components may have to be offloaded to the cloud [12]. Deciding which components to offload to the cloud and which ones to host in the edge data center is a complex optimization problem, in which capacity, latency, and data protection constraints have to be satisfied, while costs stemming from using the cloud and from data transfer between the edge data center and the cloud are to be minimized. Thus, the optimization problem we solve entails the following concerns. First, using commercial cloud services and transferring data between the edge data center and the cloud may incur financial costs. Second, components requiring low-latency communication with end devices may have to remain in the edge data center to satisfy the latency requirements. Third, components dealing with sensitive data may be prohibited to be offloaded to a public cloud due to data protection reasons [13].

Optimization is not a one-off activity. The deployment should be re-optimized during operation, e.g., when a new application is added or an application is removed, the load on an application changes, cloud prices change, etc. After such events, it may be beneficial to offload some components from the edge data center to the cloud or vice versa. To facilitate such *dynamic re-optimization*, the optimization algorithm has to be fast enough to be used online.

Novelty. The addressed problem is different from the application placement problem [14], faced by *application managers* aiming to optimally deploy their applications on a set of edge and cloud resources. In contrast, our problem is faced by *operators of edge data centers* aiming to optimally use their edge data centers' resources while satisfying the requirements of deployed applications.

Most existing approaches for application placement in fog computing are *not directly applicable* to this problem, because of different limitations (see Sec. 8 for details). On the one hand, most approaches do not account for financial costs (in particular of data transfers between the edge and the cloud) or are limited to applications of a given structure. On the other hand, existing approaches apply either simple

This paper has been accepted for publication in IEEE Transactions on Services Computing, *https://doi.org/10.1109/TSC.2022.3144645*

- *Z. Á. Mann, A. Metzger, and K. Pohl are with the University of Duisburg-Essen.*
- *J. Prade and R. Seidl are with Nokia.*

greedy algorithms with no quality guarantees, or general-purpose mathematic programming methods like integer programming that exhibit scalability issues.

Contribution. We make the following contributions:

- We formalize the problem of deciding which components to place in the edge data center and which ones to offload to the cloud, taking into account capacity, latency, and data protection constraints, while minimizing financial costs.
- We prove that the problem is strongly NP-hard.
- We devise a heuristic algorithm (FOGPART) for the problem. FOGPART exploits the graph-theoretic structure of the specific problem and can thus find good solutions quickly.
- We prove that the result of FOGPART always satisfies the capacity, latency and data protection requirements, whenever this is possible.
- We show how FOGPART can be extended for the decentralized management of multiple edge data centers and for the optimization of end-to-end application latency.

Results. We demonstrate the applicability of our algorithm by applying it to a *smart manufacturing use case*. We *experimentally evaluate* the effectiveness of FOGPART in terms of the resulting financial costs and the algorithm's execution time. The results show that FOGPART outperforms two typical types of application placement algorithms: FOGPART is faster than a typical algorithm based on integer programming, and FOGPART delivers better results than a typical greedy algorithm. The cost of the deployment found by FOGPART is on average only 2.7% higher than the results of the integer programming algorithm. However, FOGPART is orders of magnitude faster, taking less than 300ms on a commodity computer to optimize the deployment of 450 components. Thus, FOGPART delivers near-optimal results very quickly, making it applicable to practical use.

Further information. A preliminary version of this paper appeared in [15]. Since then, we extended the optimization problem and enhanced FOGPART to solve this extended problem. We evaluated the enhanced algorithm by an additional set of experiments, performed a theoretical analysis of the problem and proved the correctness of FOGPART. We also provide two novel extensions of FOGPART for the decentralized management of multiple edge data centers.

Next, Sec. 2 presents the "Factory in a Box" use case to motivate our research. Sec. 3 defines the investigated problem. Sec. 4 describes the FOGPART algorithm, while Sec. 5 provides a rigorous analysis of the algorithm's time complexity and correctness. Sec. 6 illustrates the operation of the algorithm on the case study, followed by the results of controlled experiments in Sec. 7. Related work is analyzed in Sec. 8 and Sec. 9 concludes the paper. Proofs, baseline algorithms, and a detailed discussion of limitations and enhancements can be found in the supplemental material.

2 A MOTIVATING EXAMPLE

We consider a smart manufacturing use case called "Factory in a Box" (FiaB). FiaB is an innovative factory solution, integrating a complete production environment in a standard 20-feet freight container (see Fig. 1a). It can host different types of production lines, such as electronic device manufacturing (see Fig. 1b). FiaB accommodates a heterogeneous internal communication infrastructure, including mobile and fixed telecommunication technologies (e.g., private LTE and 5G) to serve various Industrial IoT applications. The FiaB contains various end devices, like a 3D printer, a robot, special glasses for virtual or augmented reality, and sensors (e.g., temperature, humidity, impact sound, and particle sensors). The FiaB features an edge data center with up to 28 CPU cores, offering computing resources that can host application components. The FiaB also connects to a remote cloud infrastructure using a public network.

(a) Outside view (b) Inside view

Fig. 1: Factory in a Box (FiaB)

The applications to control the manufacturing operations of the FiaB consist of several components. The deployment of these components must respect multiple **constraints**:

- **Latency**. There are pairs of components, or pairs of a component and an end device, that must exchange data with each other with low latency. For example, the latency between the "Robot control" software component and the robot must not exceed 5ms.
- **Data protection**. Some components store or process sensitive data that must not be offloaded to the cloud. E.g., for manufacturing a product for a specific customer (lot-size-one production), personal data of the customer is stored. To comply with data protection regulations, components storing or processing such data must be protected.
- **Capacity**. The computing resources available in the edge data center of the FiaB are limited. In particular, CPU capacity is a limiting factor.

It is desirable to deploy as much as possible of the components to the edge data center in the FiaB, so as to utilize the available resources and minimize the financial costs associated with using the cloud. The company operating FiaB has full control over the edge data center, and sensitive data has to be processed in this trusted domain.

The FiaB can be **dynamically re-configured** to perform different manufacturing tasks. Therefore, new applications may need to be deployed or existing applications removed on the fly. Moreover, the deployment may be affected by other kinds of changes, e.g., failure of a device, changes in cloud prices, or load fluctuations.

Fig. 2 shows an example application deployment using the FiaB and the cloud. It can be seen that all components with data protection requirements are in the FiaB. The "Robot control" component is also in the FiaB to allow low-latency data exchange with the robot.

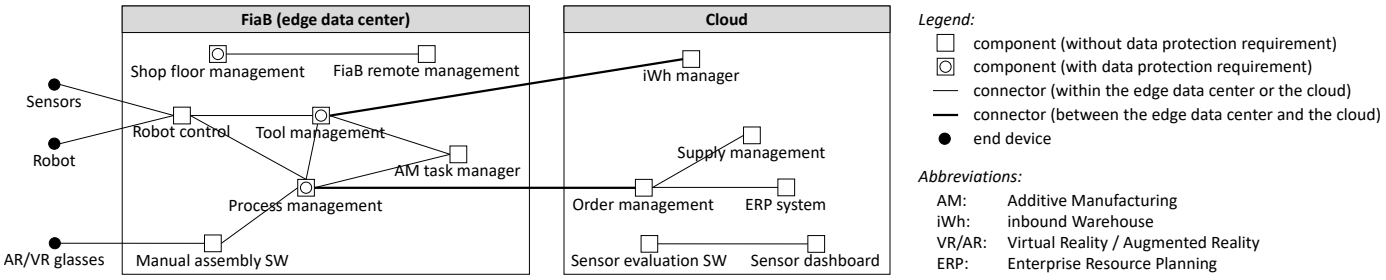


Fig. 2: An example of application components placed in the FiaB and the cloud

TABLE 1: Notation overview

Notation	Explanation
$\mathcal{A}$	Set of applications
V_A	Set of components of application A
E_A	Set of connectors among components of application A
V_D	Set of end devices connected to the edge data center
E_D	Set of connectors between components and end devices
V	Set of all components and end devices
E	Set of all connectors
$p(v)$	Processing capacity required by component v
$s(v)$	True iff component v processes sensitive data
$h(e)$	Amount of data exchange through connector e
$\ell(e)$	Maximal allowed latency for connector e
P	Processing capacity available in the edge data center
L	Latency between the edge data center and the cloud
c_p	Unit cost of processing resources in the cloud
c_dt	Unit cost of data transfer between edge data center and cloud
d	Deployment function
edge	Label for components placed in the edge data center
cloud	Label for components placed in the cloud
$\varrho(d)$	Processing capacity used in the edge data center
$E(d)$	Set of connectors between the edge data center and the cloud
$\mathrm{cost}(d)$	Financial costs of deployment d
F	Set of critical components and end devices

3 Problem Description

We first define the addressed optimization problem (see also Table 1) and then provide a theoretical analysis of the solvability and complexity of the problem.

3.1 Formal problem definition

The set of *applications* is denoted by $\mathcal{A}$. Each application $A \in \mathcal{A}$ is represented by an undirected graph (V_A, E_A), where V_A is the set of *components* of application A and E_A is the set of *connectors* among the components. V_D denotes the set of *end devices* connected to the edge data center, and E_D is the set of connectors between end devices and components. The set of all end devices and components (jointly referred to as *vertices*) is $V = V_\mathrm{D} \cup \bigcup\{V_A : A \in \mathcal{A}\}$. The set of all connectors between end devices and components as well as among components is $E = E_\mathrm{D} \cup \bigcup\{E_A : A \in \mathcal{A}\}$.

For a component $v \in V$, $p(v) \in \mathbb{R}^+$ is the compute capacity (e.g., number of CPU cores or CPU frequency) required by v. Predicate $s(v)$ is true if and only if v processes sensitive data and must hence be in the edge data center. For a connector $e \in E$, $h(e) \in \mathbb{R}^+$ is the amount of data exchanged along e, and $\ell(e) \in \mathbb{R}^+$ is the maximum allowed latency for e. To handle end devices and components uniformly, we extend the definition of p and s for end devices. For an end device $v \in V_\mathrm{D}$, $p(v) = 0$ and $s(v) = \text{true}$.

$L \in \mathbb{R}^+$ denotes the latency between the edge data center and the cloud. $P \in \mathbb{R}^+$ denotes the compute capacity (e.g., number of CPU cores or CPU frequency) of the edge data center. The cost of renting a processing unit (e.g., one vCPU) in the cloud is denoted by c_p, the unit price of data transfer between the edge data center and the cloud by c_dt.

A *deployment* is a function $d : V \to \{\text{edge}, \text{cloud}\}$ that maps each component[1] to either the edge data center or the cloud. We use $\varrho(d)$ to denote the total compute capacity used in the edge data center by deployment d:

$$\varrho(d) = \sum_{v \in V,\ d(v)=\text{edge}} p(v).$$

A *valid* deployment respects the following constraints:

$$\varrho(d) \leq P, \tag{1}$$

$$\forall v \in V :\ s(v) \Rightarrow (d(v) = \text{edge}), \tag{2}$$

$$\forall vw \in E :\ (\ell(vw) < L) \Rightarrow (d(v) = d(w)). \tag{3}$$

(1) ensures that the total processing power required by components allocated to the edge data center does not exceed its capacity. (2) ensures that all components dealing with sensitive data are deployed to the edge data center. (3) ensures that the connectors' latency requirements are met.

Our aim is to find a valid deployment that minimizes financial cost. For deployment d, the set of connectors between the edge data center and the cloud is $E(d) = \{uv \in E : d(u) \neq d(v)\}$. The *cost* of d is defined as follows:

$$\mathrm{cost}(d) = \sum_{v \in V,\ d(v)=\text{cloud}} c_\mathrm{p} \cdot p(v) + \sum_{e \in E(d)} c_\mathrm{dt} \cdot h(e), \tag{4}$$

where the first term is the total cost of leased compute resources in the cloud, and the second term is the total cost of data transfers between the edge data center and the cloud. (Recall that we address the problem from the perspective of the edge data center provider, for which using the resources in the edge data center does not incur costs.)

For an end device $v \in V_\mathrm{D}$, we defined $s(v)$ to be true. As a consequence of (2), this implies $d(v) = \text{edge}$. If end device v is connected to a component w, then, because of (3), w can only be deployed to the cloud if $\ell(vw) \geq L$. If w is deployed to the cloud, then $vw \in E(d)$ and hence the data transfer along the connector vw contributes to $\mathrm{cost}(d)$.

The *Minimum-Cost Edge-Cloud Deployment (MCECD) problem* consists of minimizing (4) while satisfying (1)-(3).

1. To simplify the problem formulation, d is also defined for end devices. As we will see, $d(v) = \text{edge}$ for any end device v, in accordance with the fact that end devices cannot be "offloaded" to the cloud.

3.2 Solvability

To analyze under which conditions the MCECD problem is solvable, we first introduce some notions.

Definition 1. *Connector $e \in E$ is* critical *if $\ell(e) < L$.*

Remark 2. *According to (3), if $vw \in E$ is critical, then for any valid deployment, either both v and w must be in the edge data center or both must be in the cloud.*

Definition 3. *A component $v \in V$ is* critical *if and only if*

- *$s(v)$ is true; or*
- *There is a path $v_0, v_1, \ldots, v_k$ in the graph (V, E), such that $s(v_0) = \text{true}$, $v_k = v$ and for each $j = 1, \ldots, k$, the connector $v_{j-1}v_j$ is critical.*

Each end device is also considered to be critical. Let $F = \{v \in V : v \text{ is critical}\}$ be the set of critical components and end devices.

Proposition 4. *Let $v \in V$ be critical. Then for any valid deployment d, $d(v) = \text{edge}$.*

(The proofs of all propositions and theorems can be found in the supplemental material.)

(1)-(3) may lead to a contradiction if the edge data center does not have enough capacity to host all critical components, but otherwise, the constraints are satisfiable:

Proposition 5. *A valid deployment exists if and only if*

$$\sum_{v \in F} p(v) \leq P. \tag{5}$$

If (5) holds, the following deployment is valid:

$$d(v) = \begin{cases} \text{edge}, & \text{if } v \in F; \quad (6a) \\ \text{cloud}, & \text{otherwise.} \quad (6b) \end{cases}$$

Proposition 5 yields a necessary and sufficient condition for the solvability of the MCECD problem, which can be checked in linear time. In the following, we assume that (5) holds so that a valid deployment exists, and we can focus on finding the solution with the minimum costs.

3.3 Complexity Analysis

The NP-hardness of problems similar to MCECD was often claimed in the literature [16], [17], [18], but seldom proven. Even if similar problems are NP-hard, this does not imply NP-hardness of MCECD. We prove a stronger claim: MCECD is strongly NP-hard, i.e., it is NP-hard even in the special case when all numbers in the problem are polynomially bounded with respect to the problem size [19].

Theorem 6. *The MCECD problem is strongly NP-hard.*

As a consequence of strong NP-hardness, we cannot expect a polynomial-time, nor even a pseudo-polynomial-time exact algorithm, nor a fully polynomial-time approximation scheme for this problem, under standard assumptions of complexity theory [19]. Thus, the MCECD problem is more complex than the related Knapsack or similar packing problems. The increased complexity stems from the graph structure and the costs of data transfer in the MCECD problem. This is why we base our approach (presented in Sec. 4) on an algorithm for minimum-cost graph partitioning.

Algorithm 1 Adding an application

```
1: procedure ADD(A)
2:     for v ∈ V_A do
3:         if s(v) then
4:             d(v) ← edge
5:         else
6:             d(v) ← cloud
7:         end if
8:     end for
9:     RE-OPTIMIZE(d)
10: end procedure
```

Algorithm 2 Removing an application

```
1: procedure REMOVE(A)
2:     for v ∈ V_A do
3:         remove v
4:     end for
5:     RE-OPTIMIZE(d)
6: end procedure
```

Algorithm 3 Handling changes

```
1: procedure CHANGES
2:     for v ∈ V do
3:         if s(v) and d(v) = cloud then
4:             d(v) ← edge
5:         end if
6:     end for
7:     RE-OPTIMIZE(d)
8: end procedure
```

3.4 Transformation

We now describe a transformation of the input of the MCECD problem, which can be used as a preprocessing step before any algorithm is applied to solve the problem. The transformation reduces the number of different kinds of constraints that have to be taken into account. The idea of the transformation is to coalesce critical connectors. Coalescing a connector $uv \in E$ means that u and v are merged to a single new vertex w. Connectors that were incident to u or v are now incident to w. The old connector uv is removed. We define $p(w) = p(u) + p(v)$ and $s(w) = s(u) \vee s(v)$. This procedure is repeated for each critical connector. An example can be found in the supplemental material.

According to Remark 2, such a coalescing step does not influence the solvability of the MCECD problem, since u and v must be deployed together anyway (either both in the edge data center or both in the cloud). Because of the definition of $p(w)$, the cost of valid deployments is also not affected by the coalescing.

When all critical connectors have been coalesced, the latency requirements are already ensured and do not have to be taken into account explicitly anymore. A further consequence is that, after the transformation, we have $s(v) = \text{true}$ for each critical vertex v. In the following, we assume that this transformation has been carried out.

4 The FogPart algorithm

We first give an overview of the main steps of the proposed algorithm, followed by a detailed description of its core.

4.1 Overview

The proposed FogPart algorithm tentatively allocates and moves the components between the edge data center and the cloud in an internal model, without an immediate effect on the real deployment. After the algorithm terminates, the best found deployment is enacted by actually carrying out the necessary allocations and migrations.

The deployment is adapted in three cases: (i) when an application is added, (ii) when an application is removed, (iii) when something changes in the deployed applications or their environment. When an application is added, we deploy each component v of the new application with the rules given in (6a)-(6b), and then re-optimize the deployment (see Algorithm 1). When an application is removed, we remove all its components from the deployment, and then perform re-optimization (see Algorithm 2). When there is a change in the deployed applications (e.g., in the CPU requirements of some components) or the infrastructure (e.g., in the unit price of using cloud resources), we first ensure that still all critical components are placed in the edge data center, and then perform re-optimization (see Algorithm 3).

Re-optimization is performed in the same way in each of the three cases. Re-optimization is based on iterative improvement, i.e., it starts from a – not necessarily valid – deployment and tries to improve it (i.e., making it valid and decreasing its cost) through a series of local changes. In each step, one component is moved either from the edge data center to the cloud or vice versa. The algorithm always makes the move that seems best, in the following sense:

- If the current deployment violates the capacity constraint, then only moving a component from the edge data center to the cloud is considered.
- From the possible moves, the one that leads to the highest decrease in deployment cost is selected.

The quantity that forms the basis for decision-making is called the *gain* of the components and is defined as follows.

Definition 7. *Let d be a deployment and $v \in V$ a component. Let d' be the deployment obtained from d by moving v. That is, for a component $w \in V$,*

$$d'(w) = \begin{cases} d(w) & \text{if } w \neq v, \\ \mathit{edge} & \text{if } w = v \text{ and } d(v) = \mathit{cloud}, \\ \mathit{cloud} & \text{if } w = v \text{ and } d(v) = \mathit{edge}. \end{cases}$$

Then, given deployment d, the gain of moving v is defined as

$$\mathit{gain}(d, v) = \begin{cases} -\infty & \text{if } d \text{ is valid, } d' \text{ is invalid}, \\ \mathit{cost}(d) - \mathit{cost}(d') & \text{otherwise}. \end{cases}$$

The algorithm makes the move with highest gain, even if this gain is negative, i.e., the cost increases (except if the gain is $-\infty$). Thus, the algorithm can *leave a local optimum* by a worsening move, hoping to unlock cost reduction opportunities that compensate the worsening, leading to a better solution in the end. To avoid infinite loops, each component may be moved only once. When no further move is possible, the valid deployment with the lowest cost encountered is taken as the resulting new deployment.

The re-optimization procedure used in FOGPART is an extended version of the Kernighan-Lin (KL) algorithm for balanced graph partitioning [20]. The KL algorithm and its variants have been successfully applied to different partitioning problems. The KL algorithm is a fast heuristic that can escape local optima. Applying the KL algorithm to our problem required several extensions, since the original algorithm supports only edge costs, whereas our problem also contains costs related to vertices, as well as hard constraints on capacity and on the placement of critical components, which are also not supported by the original algorithm.

Algorithm 4 Deployment re-optimization

```
 1: procedure RE-OPTIMIZE(d)
 2:     best_deployment ← d
 3:     best_cost ← cost(d)
 4:     L ← {v ∈ V : ¬s(v)}
 5:     end ← (L = ∅)
 6:     while ¬end do
 7:         best_gain ← −∞
 8:         for v ∈ L do
 9:             if ϱ(d) ≤ P or d(v) = edge then
10:                 g ← GAIN(d,v)
11:                 if g > best_gain then
12:                     best_comp ← v
13:                     best_gain ← g
14:                 end if
15:             end if
16:         end for
17:         if best_gain > −∞ then
18:             forced ← (ϱ(d) > P)
19:             change d(best_comp) to the other value
20:             L.remove(best_comp)
21:             if forced or cost(d) < best_cost then
22:                 best_deployment ← d
23:                 best_cost ← cost(d)
24:             end if
25:         end if
26:         end ← (L = ∅ or best_gain = −∞)
27:     end while
28:     d ← best_deployment
29: end procedure
```

4.2 Detailed description of deployment re-optimization

The re-optimization procedure is shown in Algorithm 4. The algorithm starts by setting "best_deployment" and "best_cost" to the current deployment respectively its cost (lines 2-3). The list L contains the components that may be moved. In line 4, L is initialized to the set of all non-critical components; critical components are not movable since they must remain in the edge data center. In each iteration, one component is moved and it is removed from L (line 20); the procedure ends if L becomes empty, as captured by the Boolean variable "end" (lines 5, 6, 26).

In each iteration, the component to move ("best_comp") is determined. For this, "best_gain" is initialized to $-\infty$ (line 7), and all movable components are checked (lines 8-16). Moving a component from the cloud to the edge data center is not considered if the edge data center is overloaded (Line 9). Lines 10-14 find the component with the highest gain. If an allowed move is found, it is performed (line 19) and the corresponding component is removed from L (line 20). If the edge data center was overloaded before the move, then the move is forced to be from the edge data center to the cloud, as captured by the Boolean variable "forced". In this case, "best_deployment" and "best_cost" are certainly updated with the new deployment and its cost, otherwise they are updated only if the new deployment has lower cost than the best deployment found so far (lines 18, 21-24). The loop ends if there are no more movable components ($L = \emptyset$) or there are no valid moves, i.e., there are only moves that would invalidate the deployment ("best_gain" $= -\infty$) (line 26). Finally, the best deployment found is chosen (line 28).

The gain of a component is computed by Algorithm 5, in line with Definition 7. If component v is in the edge data center, then moving it to the cloud would increase costs by $c_{\mathrm{p}} \cdot p(v)$ (lines 2-3); otherwise, moving it to the edge data

Algorithm 5 Calculation of the gain of moving a component

```
1:  procedure GAIN(d, v)
2:      if d(v) = edge then
3:          r ← −c_p · p(v)
4:      else if ϱ(d) ≤ P and ϱ(d) + p(v) > P then
5:          return −∞
6:      else
7:          r ← c_p · p(v)
8:      end if
9:      for vw ∈ E do
10:         if d(v) = d(w) then
11:             r ← r − c_dt · h(vw)
12:         else
13:             r ← r + c_dt · h(vw)
14:         end if
15:     end for
16:     return r
17: end procedure
```

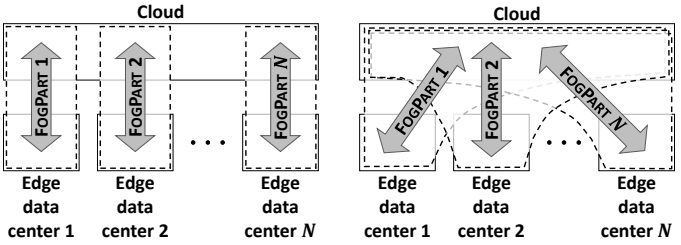
center would decrease costs by the same amount (lines 6-7). However, if the move violates the capacity constraint of the edge data center, then the move is not allowed, resulting in a gain of $-\infty$ (lines 4-5). In lines 9-15, the connectors incident to v are investigated. For a connector vw, if v and w are deployed the same way (either both are in the edge data center or both are in the cloud), then the move results in vw crossing the boundary between the edge data center and the cloud, increasing costs by $c_{\text{dt}} \cdot h(vw)$ (lines 10-11). If one of the components is in the edge data center and the other one in the cloud, then moving v results in vw not crossing the boundary between edge data center and cloud anymore, thus decreasing costs by the same amount (lines 12-13).

4.3 Extension to multiple edge data centers

FOGPART, as described above, manages a single edge data center, such as in the scenario described in Sec. 2. Here, we present an extension of this algorithm to handle situations with multiple edge data centers [21], [22].

This leads to the following variant of the initial problem model. We are given a set of edge data centers and a cloud. Each edge data center has given (possibly different) computational capacity. The capacity of the cloud is assumed to be unlimited (as in the original problem formulation). Each edge data center belongs to a provider; a provider may have multiple edge data centers. Each application has a primary target edge data center, which has direct connection to the end devices used by the application. Critical components of the application must be placed on the primary target edge data center. This guarantees that connectors to end devices have the required low latency and that sensitive data is not sent through the network. Components can be placed without incurring costs on the primary target edge data center or any other edge data center of the same provider. Further nodes (edge data centers or the cloud) can also be used, but they are associated with given (possibly different) costs. Data transfer between each pair of nodes is associated with given (possibly different) costs. The objective is to minimize the total costs of the rental of computational capacity plus the costs of data transfer among nodes, while satisfying the constraints stemming from critical components and from the edge data centers' capacity.

For solving such problems, so far mainly *centralized* approaches have been proposed [23]. In these approaches, one



(a) DISTFOGPART (b) CROSSFOGPART

Fig. 3: Handling multiple edge data centers

entity collects information about the whole infrastructure and all applications, makes decisions on optimizing application placement, and then sends adaptation commands to the involved nodes. Such centralized approaches offer limited scalability, suffer from the risk of the single point of failure, and may not be applicable in practical situations involving multiple autonomous operators. Therefore, a key challenge for managing such distributed settings is to develop *decentralized* algorithms that can work on each node independently, with as little coordination as possible [23].

We propose two ways to extend FOGPART for the decentralized management of multiple edge data centers. In the first approach, called DISTFOGPART, each edge data center runs an independent instance of the FOGPART algorithm (Fig. 3a). That is, FOGPART instance i manages edge data center i and the applications targeted to edge data center i, and decides which components of those applications should be placed in edge data center i and which ones in the cloud. This requires no modification to FOGPART, and no coordination among the FOGPART instances.

In the second approach, called CROSSFOGPART, each edge data center runs an instance of FOGPART and each FOGPART instance may change the placement of any component currently placed in the cloud (Fig. 3b). Thus it is possible that FOGPART instance i places a component c of an application targeted to edge data center i in the cloud, and then c is migrated by FOGPART instance j to edge data center j. Hence, CROSSFOGPART supports the optimized placement of an application even across multiple edge data centers. To avoid concurrent conflicting modifications of the contents of the cloud by multiple FOGPART instances, a lightweight synchronization among the FOGPART instances is necessary. In our current proof-of-concept implementation, this is realized by a central orchestrator, which ensures that, after one FOGPART instance made a modification (e.g., added a new application), all other FOGPART instances also perform re-optimization one after the other. To calculate communication costs, a FOGPART instance may also need information about the placement of components on other edge data centers, which is currently also provided by the central orchestrator. As future work, the central orchestrator may be replaced by a decentralized data handling and coordination mechanism (e.g., a distributed locking protocol).

4.4 Extension to global latency optimization

The literature contains different interpretations of latency. Some authors define latency as the total delay on a path

or cycle of the application graph [24]; others define latency constraints for individual connectors [18], [25], [26]. Our approach, as described so far, belongs to this second category.

Our approach can be extended to take into account the end-to-end latency of applications. We assume that for each application $A \in \mathcal{A}$, a set of connectors E_A^{lat} is given (e.g., the connectors forming a path or a cycle), the total latency of which, denoted as T_A, should be minimized. We then modify the objective function in our problem definition (equation (4)) as follows: minimize $\text{cost}(d) + \lambda \cdot \sum_{A \in \mathcal{A}} T_A$. Here, $\lambda \geq 0$ is a given constant, representing the relative importance of end-to-end latency to financial costs.

To extend FOGPART accordingly, the cost calculation (lines 3, 21, 23 in Algorithm 4) and gain calculation (lines 9-15 in Algorithm 5) need to be changed. In the gain calculation, changes to the latency of connectors contained in E_A^{lat} have to be considered. The resulting algorithm is denoted as Global Latency Optimization with FOGPART, or GLOFOGPART for short.

5 ANALYSIS

In this section, we analyze the computational complexity of the FOGPART algorithm and prove its correctness.

Complexity. FOGPART has quadratic time complexity and linear space complexity:

Theorem 8. *The time complexity of Algorithm 4 is* $O\big(|V| \cdot (|V| + |E|)\big)$.

Corollary 9. *The time complexity of Algorithms 1, 2, and 3 is also* $O\big(|V| \cdot (|V| + |E|)\big)$.

Theorem 10. *Algorithm 5 requires* $O(1)$ *auxiliary space. Algorithms 1–4 require* $O(|V|)$ *auxiliary space.*

Correctness. To prove the correctness of our algorithms, we reason about sequences of algorithm calls.

Definition 11. *A* call sequence *is a list* $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_k)$*, where each* $\gamma_i \in \{\textit{add}, \textit{remove}, \textit{change}\}$*, depending on whether the* i*th call was to Algorithm 1 (adding an application), Algorithm 2 (removing an application), or Algorithm 3 (other change). The set of applications and the deployment after* i *calls are denoted by* $\mathcal{A}^{(i)}$ *and* $d^{(i)}$*, respectively. In particular,* $\mathcal{A}^{(0)}$ *and* $d^{(0)}$ *denote the set of applications respectively the deployment before the first call.*

Theorem 12. *Performing an arbitrary call sequence starting from* $\mathcal{A}^{(0)} = \emptyset$*, if condition (5) is satisfied throughout (i.e., the problem is solvable), then each call results in a valid deployment.*

Thus, our algorithms always return deployments that **satisfy all constraints**, whenever this is possible.

6 CASE STUDY

To demonstrate the applicability of FOGPART and illustrate its operation, we apply it to the FiaB use case from Sec. 2.

For managing the production in the FiaB, multiple applications are used. Table 2 shows the application components' characteristics: the 1st column contains the application identifier, the 2nd column contains the component's name (cf. the abbreviations from Fig. 2), the 3rd column shows the component's CPU requirement $p(v)$ (number of required vCPUs), and the 4th column indicates if the component is subject to data protection requirements ($s(v)$). Table 3 shows the connectors' characteristics: the 1st column contains the application identifier, the 2nd column shows the vertices incident to the connector, and the 3rd column shows the amount of data transfer $h(e)$ along the connector in GB/day. The first three connectors connect an end device with a component; the other connectors connect two components.

TABLE 2: Characteristics of components in the case study

App.	Component	vCPUs	Data protection
A_1	AM task manager	1	no
	iWh manager	1	no
	Robot control	1	no
	Manual assembly SW	2	no
	Order management	2	no
	Supply management	2	no
	Tool management	1	yes
	Process management	1	yes
	ERP system	2	no
A_2	FiaB remote management	1	no
	Shop floor management	1	yes
A_3	Sensor evaluation SW	1	no
	Sensor dashboard	1	no

TABLE 3: Characteristics of connectors in the case study

App.	Connector (endpoint1 ↔ endpoint2)	Data
A_1	AR/VR glasses (device) ↔ Manual assembly SW	15
	Sensors (device) ↔ Robot control	2
	Robot (device) ↔ Robot control	0.5
	Tool management ↔ Process management	1
	AM task manager ↔ Tool management	2
	iWh manager ↔ Tool management	0.5
	Robot control ↔ Tool management	2
	AM task manager ↔ Process management	0.1
	Robot control ↔ Process management	0.1
	Manual assembly SW ↔ Process management	1
	ERP system ↔ Order management	1
	Order management ↔ Supply management	0.1
	Order management ↔ Process management	0.1
A_2	Shop floor management ↔ FiaB remote management	5
A_3	Sensor evaluation SW ↔ Sensor dashboard	2.5

For real-time robot control, the communication latency between (i) the "Sensors" device and the "Robot control" component and (ii) between the "Robot control" component and the "Robot" device must not exceed 5 milliseconds. Hence, for these two connectors we specify $\ell(e) = 5\text{ms}$. The other connectors use loosely-coupled, asynchronous communication with no specific latency requirements; thus, for all other connectors we set $\ell(e) = \infty$.

The unit costs of compute resources in the cloud and of data transfers between the edge data center and the cloud are determined based on Amazon EC2 pricing[2]. The hourly rental fee of a t2.small instance is 0.023\$, leading to a daily fee of $c_{\text{p}} = 0.552\$$. The transfer of 1GB of data to or from Amazon EC2 costs $c_{\text{dt}} = 0.09\$$. The edge data center has $P = 12$ CPU cores. The latency between the edge data center and the cloud is $L = 100\text{ms}$.

As the connectors between the "Robot control" component and the "Sensors" and "Robot" devices have a lower latency requirement than the latency between the edge data center and the cloud, "Robot control" is a critical component. Thus, "Robot control" must be in the edge data center, just as all components with data protection requirements.

2. https://aws.amazon.com/ec2/pricing/on-demand/

(a) Result of deploying the first application

(b) Deployment of the second application. The numbers show the order in which the components are allocated and moved by FOGPART. The two numbers next to "FiaB remote management" show that this component is allocated in the first step (at that time in the cloud), and moved in the fourth step (to the edge data center)

(c) Deployment of the third application. The numbers show the order in which the components are allocated and moved by FOGPART

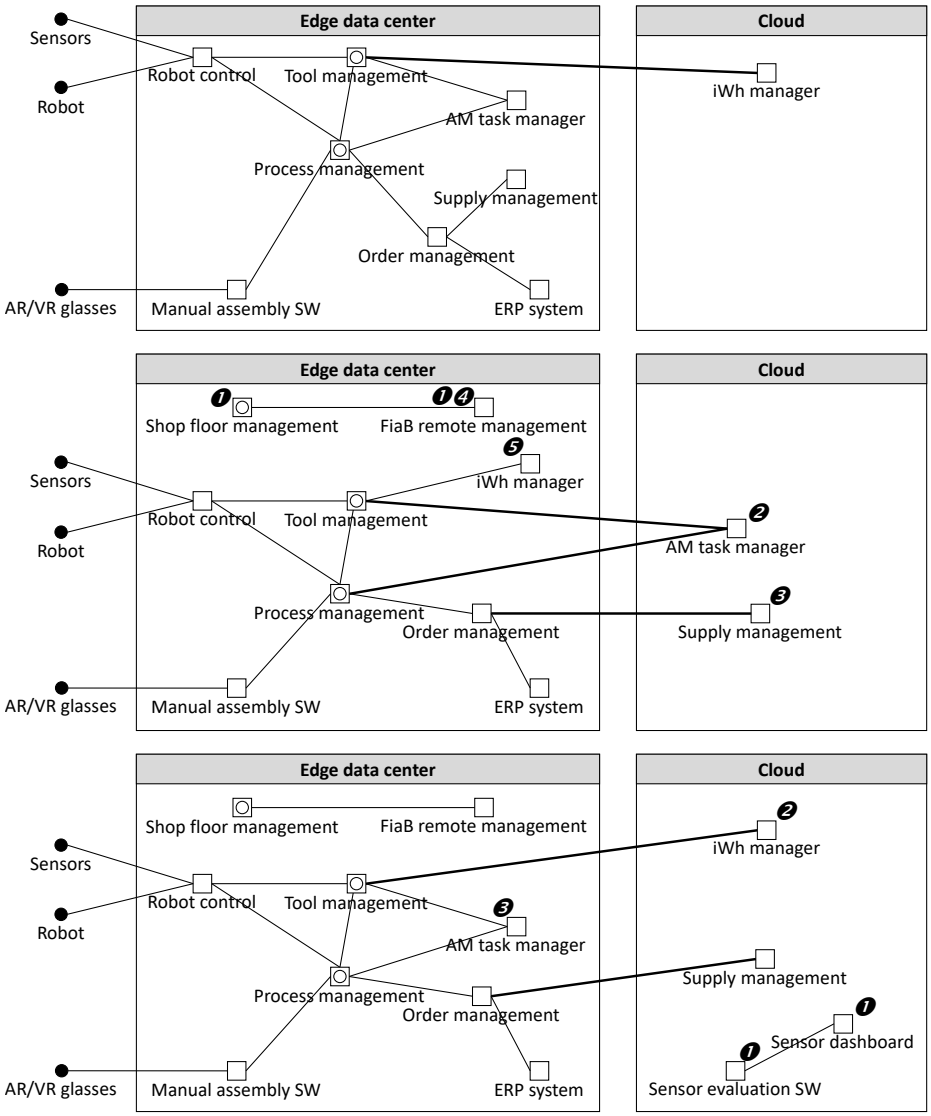


Fig. 4: Deploying the applications of the case study, one after the other. The graphical notation is the same as in Fig. 2.

Running FOGPART to add application A_1, first the critical components (Robot control, Tool management, Process management) are put into the edge data center and all other components are tentatively put into the cloud. Then, Algorithm 4 is run to optimize the deployment. Algorithm 4 moves five components from the cloud to the edge data center, until the capacity of the edge data center is exhausted, leading to the deployment shown in Fig. 4(a).

To deploy application A_2, first its critical component (Shop floor management) is put into the edge data center and the other (FiaB remote management) into the cloud. This leads to an invalid deployment requiring 13 CPU cores in the edge data center. Hence, FOGPART makes a forced move: "AM task manager" is moved from the edge data center to the cloud. Thus, the deployment becomes valid, even reaching a local optimum: only moves from the edge data center to the cloud are possible, which increase costs. FOGPART makes such a worsening move of "Supply management" from the edge data center to the cloud. This pays off: 2 CPU cores are freed in the edge data center, allowing the "FiaB remote management" and "iWh manager" components to move in the next steps to the edge data center. The resulting deployment is better than the earlier local optimum, as the heavy traffic between "Shop floor management" and "FiaB remote management" remains in the edge data center. This is an example of how FOGPART can escape local optima. In subsequent steps, FOGPART tries further moves but they do not lead to lower costs, hence the deployment shown in Figure 4(b) is chosen in the end.

For A_3, the new components (Sensor evaluation SW, Sensor dashboard) are first put in the cloud. Since the capacity of the edge data center is exhausted, only worsening moves are possible. FOGPART moves "iWh manager" from the edge data center to the cloud, making it possible to move "AM task manager" from the cloud to the edge data center. This leads to a better deployment, which further moves cannot improve. The resulting deployment, shown in Fig. 4(c), is optimal. Also, it has lower costs than the manually created deployment for the same inputs shown in Fig. 2.

This shows how FOGPART keeps satisfying the requirements, and uses the remaining degrees of freedom to optimize costs, occasionally also escaping local optima.

7 Experimental evaluation

We experimentally assess the costs of the solutions delivered by FogPart and its execution time. We compare FogPart to two other algorithms. To this end, we implemented all three algorithms in a common Java program[3]. The experiments are performed on a Lenovo ThinkPad X1 laptop with Intel Core i5-4210U CPU @ 1.70GHz and 8GB RAM[4].

7.1 Baseline algorithms

We compare FogPart to two competing algorithms:

- An algorithm based on integer linear programming (ILP), as a typical example of an exact algorithm. ILP was used by several previous works, e.g., [27].
- A heuristic based on the first-fit (FF) principle, as a typical example of a greedy heuristic. Similar heuristics were used by several previous works, e.g., [28].

These algorithms are described next. All three algorithms solve the MCECD problem after the transformation of Sec. 3.4.

Exact algorithm using integer linear programming (ILP). To create an ILP formulation of the MCECD problem, we define two sets of binary variables $\{x_v : v \in V\}$ and $\{y_e : e \in E\}$ with the following interpretation:

$$x_v = \begin{cases} 0 & \text{if } d(v) = \text{edge} \\ 1 & \text{if } d(v) = \text{cloud} \end{cases} \qquad y_e = \begin{cases} 0 & \text{if } e \notin E(d) \\ 1 & \text{if } e \in E(d) \end{cases}$$

The ILP can be formulated as follows:

$$\min \quad c_{\mathrm{p}} \cdot \sum_{v \in V} p(v) \cdot x_v + c_{\mathrm{dt}} \cdot \sum_{e \in E} h(e) \cdot y_e \tag{7}$$

$$\text{s. t.} \quad \sum_{v \in V} p(v) \cdot (1 - x_v) \leq P \tag{8}$$

$$x_v = 0 \qquad \text{if } s(v) \tag{9}$$

$$x_v - x_w \leq y_{vw} \qquad \forall vw \in E \tag{10}$$

$$x_w - x_v \leq y_{vw} \qquad \forall vw \in E \tag{11}$$

(7) corresponds to the cost function (4) defined earlier, while (8)-(9) correspond to the constraints (1)-(2) defined earlier. (10) and (11) ensure that the values of the x and y variables are consistent: if $x_v \neq x_w$, then $y_{vw} = 1$. If $x_v = x_w$, then the value of y_{vw} is not constrained; however, since y_{vw} has a positive weight in the objective function, $y_{vw} = 0$ will hold in any optimal solution of the ILP.

We use the Gurobi Optimizer[5], version 7.0.2, to solve the ILP defined above. In the experiments, Gurobi was executed in single-threaded mode with a timeout of 60 seconds.

First-fit heuristic. The FF heuristic works as follows:

1) It places all critical components into the edge data center.
2) It iterates over the remaining components, and does the following for each non-critical component v: if v still fits into the edge data center, then v is placed into the edge data center, otherwise into the cloud.

More details on both baseline algorithms can be found in the supplemental material.

3. https://sourceforge.net/p/vm-alloc/hybrid-deployment
4. The Intel NUC devices in the FiaB have comparable performance.
5. https://www.gurobi.com/

7.2 Results for a call sequence

In a first experiment, 10 applications are randomly generated with the following parameters:

- $|V_A| = 30$
- Independently for each $v \in V_A$, $p(v)$ is chosen randomly from $\{1, 2, 3, 4\}$ with uniform distribution, and $s(v)$ is set true with probability 0.1
- (V_A, E_A) is a complete graph
- Independently for each $e \in E_A$, $h(e)$ is chosen randomly from $[0.0, 3.0]$ with uniform distribution

Starting with $\mathcal{A}^{(0)} = \emptyset$, the applications are added one by one in the first 10 steps. Afterwards, 10 change steps are carried out, and finally the applications are removed one by one, leading to $\mathcal{A}^{(30)} = \emptyset$. Each change step performs one of the following actions (each with equal probability):

- For each application, increase or decrease the number of required CPU cores of 3 random components by 1.
- For each application, change a random component's being critical.
- For each application, multiply the amount of data transfer along 10 random connectors by 2 or 0.5.
- Change c_{p}, the unit cost of compute resources, by either increasing or decreasing it by 10%.

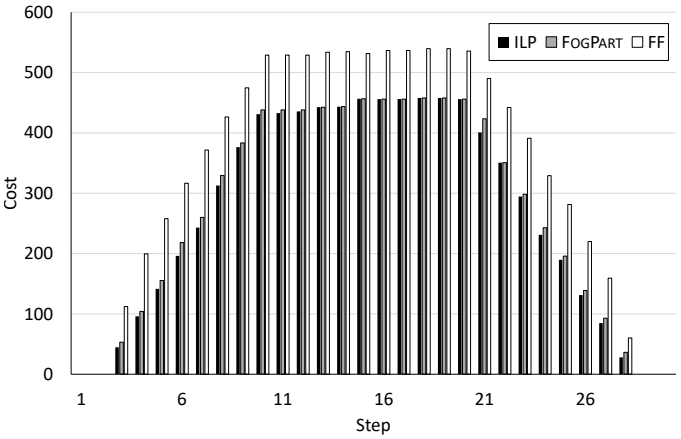
As before, $c_{\mathrm{p}} = 0.552$ and $c_{\mathrm{dt}} = 0.09$. Moreover, $P = 150$, $L = 0$, and $V_{\mathrm{D}} = \emptyset$.

Fig. 5a shows the costs achieved by the algorithms in each step. As expected, costs monotonously increase in the first 10 steps and decrease in steps 20-30. In steps 1-2 and 29-30, all components can be placed in the edge data center, leading to 0 costs; this optimal deployment is found by all algorithms. In the other steps, some components must be placed in the cloud, leading to non-zero costs. Consistently in all steps 3-28, the results of FogPart are only slightly higher than those of the ILP-based algorithm (2.19% higher on average), whereas the FF algorithm yields much higher costs (29.32% higher on average).

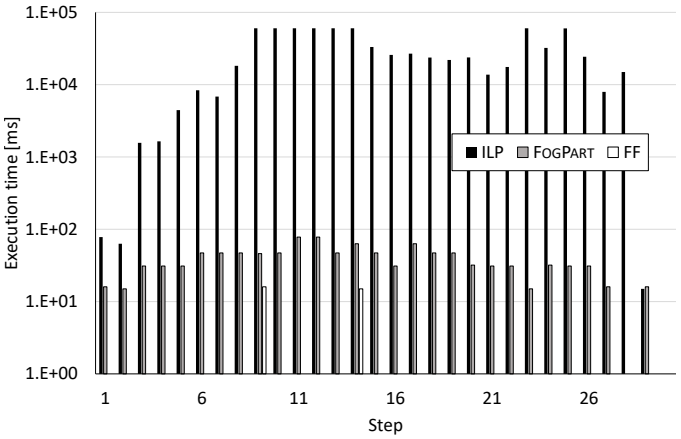
Fig. 5b shows the execution time of the algorithms in each step. Time is shown in milliseconds, using logarithmic scale. The ILP-based algorithm has consistently much higher execution time than the heuristics. The average execution time is about 26sec for the ILP-based algorithm, and only about 36ms for FogPart and 1ms for FF. In 8 cases, the ILP-based algorithm reached the 60sec timeout. In such a case, the ILP solver returns the best solution found and a lower bound on the optimal costs. In these cases, the output of the ILP-based algorithm might not be optimal; however, based on the lower bound, we can establish that the cost of the deployment found by the ILP-based algorithm is at most 0.82% higher than the optimum.

7.3 Scalability

In the next experiment, we repeated the same call sequence as in Sec. 7.2, with varying number of components per application (otherwise, all parameters are set as before). Fig. 6a shows the total costs achieved by the algorithms, aggregated for the call sequence of adding, changing, and removing 10 applications. The number of components per application increased from 15 to 45 in increments of 5, leading to 150-450 components in a call sequence.

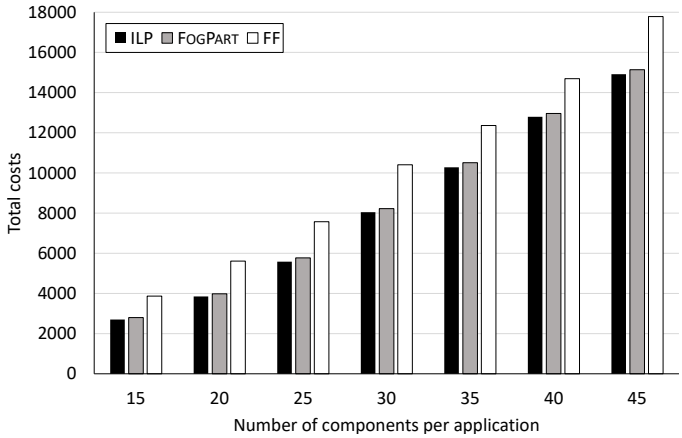


(a) Financial costs

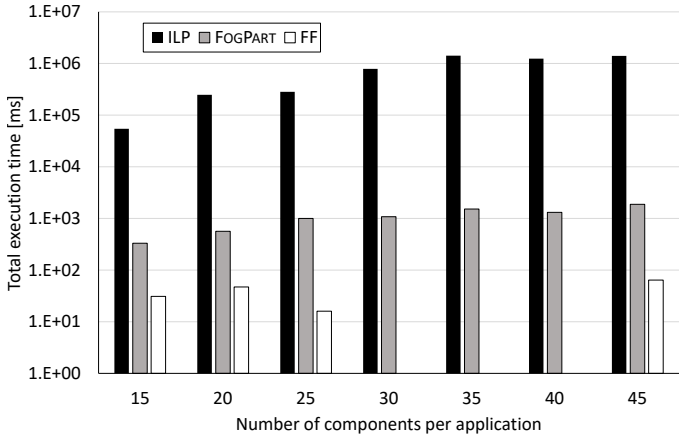


(b) Algorithm execution time (logarithmic scale)

Fig. 5: Adding, changing, and removing 10 applications



(a) Financial costs



(b) Algorithm execution time (logarithmic scale)

Fig. 6: Impact of increasing the number of components

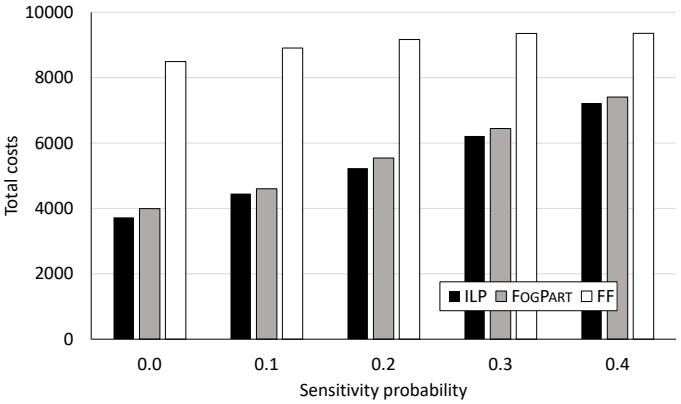
Consistently for all application sizes, the cost of the deployments found by FOGPART is only slightly higher than the ILP results (2.1% higher on average), while the costs achieved by FF are much higher (24.3% higher on average). As the number of components grows, the relative difference between the algorithms' results decreases. This is because, as the number of components grows, also the number of critical components grows, using an increasing part of the edge data center's capacity, leaving less optimization opportunities in deploying the non-critical components. E.g., when each application consists of 45 components, the expected number of CPU cores needed by the critical components is 112.5, using 75% of the capacity of the edge data center.

Fig. 6b shows the total execution time of the algorithms, aggregated for the call sequences (note the logarithmic scale of the vertical axis). The execution time of FF is very low (tens of milliseconds for the call sequence), that of FOGPART is higher but still low (less than 2s in each case for the whole call sequence), and that of ILP much higher (more than 20min for a call sequence with over 300 components). With a growing number of components, the execution time of FOGPART and ILP grow at a different rate. As the number of components triples from 150 to 450, the execution time of FOGPART increases by a factor of 5.7 (cf. the moderate polynomial complexity stated in Theorem 8), while the execution time of ILP grows by a factor of 26, exhibiting an exponential execution time damped by the timeout of 60s per run. From the 30 runs of a call sequence, the number of runs reaching the timeout is 0 for 150 components, 8 for 300 components, and 21 for 450 components.

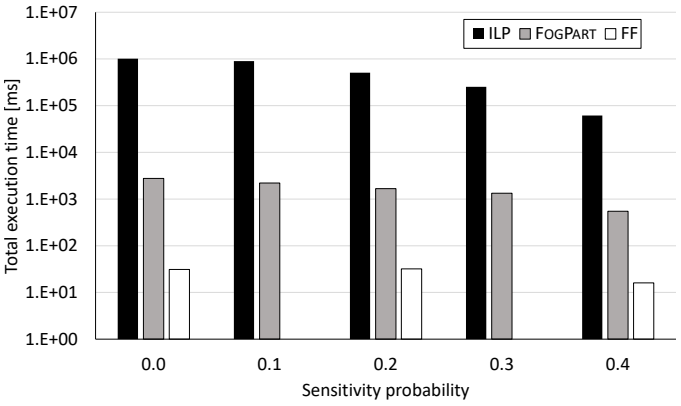
7.4 Impact of constraint tightness

In the next experiment, we varied the ratio of critical components. The probability of components being marked as processing sensitive data, i.e., Prob($s(v)$=true), varied from 0.0 to 0.4. As Prob($s(v)$=true) grows, the placement of more components is prescribed, leading to more constrained problem instances. To ensure that the problem is solvable even for Prob($s(v)$=true)=0.4, the number of CPU cores in the edge data center was increased to 350. Otherwise, all parameters were set as in the first experiment.

Fig. 7 shows the results aggregated for the call sequence of adding, changing, and removing 10 applications. The costs monotonously increase for all algorithms as Prob($s(v)$=true) increases. The difference between the results of ILP and FOGPART is always very small, and the results of FF are much worse. On average, the costs achieved by FOGPART are 4.2% higher than those of ILP; the costs achieved by FF are 68.6% worse than those of ILP. As Prob($s(v)$=true) increases, the relative difference between the algorithms' results decreases. This can be again explained by the decreasing optimization opportunities, as

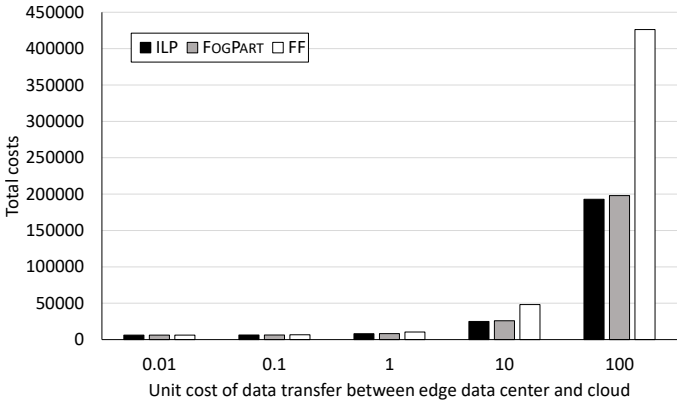


(a) Financial costs

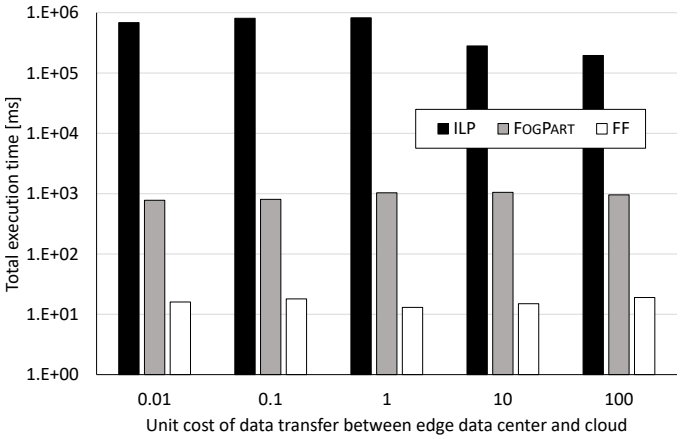


(b) Algorithm execution time (logarithmic scale)

Fig. 7: Impact of increasing the ratio of critical components



(a) Financial costs



(b) Algorithm execution time (logarithmic scale)

Fig. 8: Impact of varying c_{dt}

an increasing percentage of the edge data center's capacity is occupied by the critical components.

Fig. 7b shows the corresponding execution times (note the logarithmic scale of the vertical axis). As in the previous experiments, FF works in milliseconds, FOGPART in a few seconds, and ILP takes several minutes for a call sequence. The increase in constraint tightness leads to a decrease of the execution time of FOGPART and ILP, which may again be attributed to the decrease in optimization opportunities.

7.5 Impact of cost structure

To investigate the effect of the ratio of different costs, we performed the same experiment as in Sec. 7.2, with different values for c_{dt}, the unit cost of data transfer between the edge data center and the cloud. We multiplied the basic value of c_{dt} with different factors from 0.01 to 100, while c_{p}, the unit cost of compute resources, remained the same.

As Fig. 8a shows, the costs achieved by all algorithms monotonously increase with growing c_{dt}. The difference between the results of FOGPART and ILP is consistently very small (2.6% on average). The gap between FF and ILP is much larger (108.9% on average).

Costs grow slowly when c_{dt} is small (e.g., increasing the factor of c_{dt} from 0.01 to 0.1 results in about 2-6% increase in total costs), whereas the cost increase is much higher when c_{dt} is large (e.g., increasing the factor of c_{dt} from 10 to 100 results in over 600% increase in total costs, for each algorithm). This is because for small values of c_{dt}, overall costs are dominated by compute costs, limiting the impact of changes in communication costs. When c_{dt} is large, total costs are dominated by communication costs, so that total costs get more sensitive to changes of c_{dt}.

The results of FF deteriorate when c_{dt} is large. This is because FF does not take communication costs explicitly into account. Hence, when communication becomes expensive, FF may lead to high costs.

Fig. 8b shows the total execution time of the algorithms, aggregated for the call sequences (note the logarithmic scale of the vertical axis). As in the previous experiments, ILP is consistently multiple orders of magnitude slower than the two other algorithms. While the execution time of FOGPART and FF is hardly affected by the value of c_{dt}, the ILP-based algorithm seems to be faster for very high values of c_{dt} than otherwise. This may be attributed to the way the branch-and-bound algorithm of the ILP solver works. When c_{dt} is large, partial solutions with many connectors crossing the boundary between the edge data center and the cloud can be immediately pruned because they will surely lead to solutions with too high costs. The improved pruning capability leads to faster execution of the solver.

7.6 Results for multiple edge data centers

To evaluate DISTFOGPART and CROSSFOGPART, we compare them to each other and to applying one instance of First-Fit or ILP per edge data center. We consider 20 edge

TABLE 4: Results with multiple edge data centers

c_{dt}	$\frac{c_{ee}}{c_{dt}}$	Resulting cost			
		FF	ILP	DISTFOGPART	CROSSFOGPART
0.009	0.01	387.6	349.0	355.4	349.4
0.009	0.1	455.7	422.3	426.1	423.3
0.009	1	430.1	393.4	398.2	399.2
0.09	0.01	549.7	365.0	384.2	348.4
0.09	0.1	546.7	351.8	370.3	335.2
0.09	1	588.8	392.7	410.7	409.5
0.9	0.01	2421.1	721.0	759.6	503.4
0.9	0.1	2385.4	541.5	618.6	396.6
0.9	1	2406.0	523.2	582.8	565.3

data centers and a cloud. For each edge data center, the number of available CPU cores is randomly taken from [10, 50]. We generate 5 applications for each edge data center (i.e., 100 applications altogether). Each application consists of 5 components (i.e., we have 500 components altogether). Each component requires between 1 and 4 CPU cores, and is critical with probability 0.1. Each pair of components within an application is connected by a connector, carrying a random amount of traffic from [0, 10]. As in the previous experiments, the per-core rental fee in the cloud is 0.552. All edge data centers belong to the same provider and can thus be used free of charge. For the unit cost of data transfer between the cloud and the edge data centers (c_{dt}), we consider 0.09 as in previous experiments, and also 0.9 and 0.009, since we have seen previously that the value of c_{dt} has significant impact. For the unit cost of data transfer among edge data centers (c_{ee}), we consider the values c_{dt}, $0.1 \cdot c_{dt}$, and $0.01 \cdot c_{dt}$.

The results are shown in Table 4. In each case, both DISTFOGPART and CROSSFOGPART outperform First-Fit; in accordance with the results of Sec. 7.5, the difference is highest when c_{dt} is high. If c_{ee} is low, as can be expected in a typical edge deployment, CROSSFOGPART can further improve upon the result of DISTFOGPART by up to 34%, and in some cases even outperforms the ILP-based algorithm. This is because CROSSFOGPART can leverage spare capacity in one edge data center to cheaply host excess components from another edge data center.

7.7 Results for global latency optimization

Here, we evaluate the performance of GLOFOGPART, introduced in Sec. 4.4. Fig. 9 compares the costs achieved by GLOFOGPART with the ILP-based and the First-Fit algorithms. The cost depicted on the vertical axis is the weighted sum of the financial cost (with weight 1) and the total latency of the applications (with weight λ). The results are shown for different values of λ. The resulting costs are normalized such that the result of ILP is 1. As can be seen, with growing λ, the results of both GLOFOGPART and FF deteriorate. However, while the results of FF are up to 207% worse than those of ILP, the results of GLOFOGPART are less than 19% off the results of ILP. Thus, GLOFOGPART can effectively minimize both financial costs and total latency.

7.8 Summary

In all experiments of Sec. 7.2-7.5, the costs of the deployment found by FOGPART are only slightly higher than those of the ILP-based algorithm (2.7% higher on average across all runs in the experiments). The costs of FF are much higher, on average 88.8% higher than those of ILP.

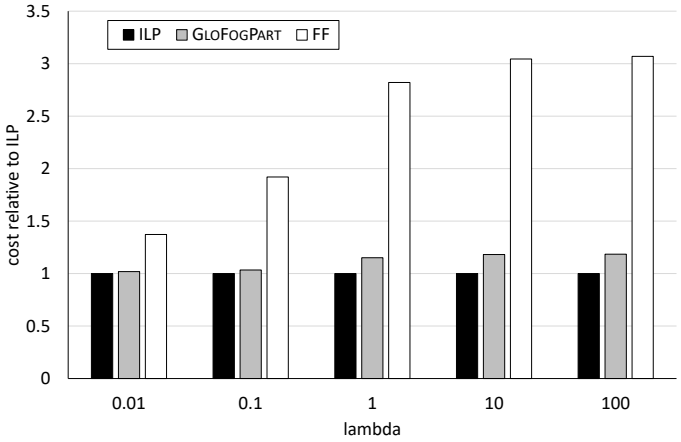


Fig. 9: Costs achieved for different values of λ

FF takes on average just 0.6ms per run. FOGPART is also very fast, taking on average 40.6ms per run, and below 300ms in each run. ILP was on average more than 500 times slower than FOGPART, taking on average about 22sec per run and hitting the 60sec timeout in many cases.

Overall, FOGPART is almost as good as ILP in terms of solution costs, while being almost as fast as FF.

The results of Sec. 7.6 show that FOGPART can be successfully applied in a setting with multiple edge data centers, leading to optimized costs with no or minimal coordination among the edge data centers. The results of Sec. 7.7 show that FOGPART can also be successfully extended to minimize overall latency. Further experimental results are presented in the supplemental material.

8 RELATED WORK

Recent surveys cover the state of the art in offloading in cloud and edge/fog computing [7], [10], [14]. Here, we focus on the approaches that are most closely related to ours.

[29] was among the first to study workload allocation between the cloud and the fog. The results showed that combining cloud and fog resources can lead to a good trade-off between delay and power consumption. [30] formulates the service allocation problem in a cloud-fog setting as an optimization problem. An ILP formulation is created and solved using Gurobi. [31] devises a genetic algorithm for optimizing the deployment of IoT applications on fog nodes. The aim is to allocate as much as possible of the service requests to the fog nodes, while satisfying constraints on capacity and response time. The algorithm is compared to an ILP-based and an FF-based approach. [28] presents a greedy algorithm for placing application modules in the cloud and on fog nodes. The algorithm ensures the satisfaction of capacity constraints, and tries to use the fog nodes as much as possible. [32] addresses the deployment of complex event processing applications on edge resources with the aim of minimizing average application latency, while satisfying capacity constraints. The algorithm is compared to a greedy algorithm and a load-balancing algorithm. [17] considers the

deployment of distributed stream processing applications on cloud and edge resources with the aim of minimizing application latency. Applications are assumed to be series-parallel-decomposable graphs. Beside capacity constraints, also the requirements on service rate are considered. [33] proposes a heuristic to allocate application components in a multi-layer fog system. The aim is to optimize resource usage while enforcing latency constraints. [34] uses tabu search to optimize the placement of virtual network functions in the context of mobile fog nodes. The optimization objective combines makespan with financial costs, while capacity constraints are ensured. [35] uses different variants of evolutionary algorithms for the fog service placement problem. The NSGA-II algorithm led to the best results, while the MOEA/D algorithm was better to reduce execution time. [36] presents a two-tier bipartite graph task allocation approach based on fuzzy clustering considering time delay constraints, costs, and energy consumption. [26] uses exhaustive search to find all feasible deployments of an application on a fog infrastructure.

Our approach differs from these previous efforts in several respects. Most existing works focus on the optimization problem faced by an application manager, whereas we address a problem faced by the operator of an edge data center. Most existing approaches are not directly applicable to our problem because of their limitations (e.g., they do not account for financial costs or are limited to applications of a given structure). In addition, most existing works applied either simple greedy algorithms without quality guarantees or general-purpose mathematic programming methods (like integer programming) that have scalability issues. In contrast, by focusing on the case of one edge data center and the cloud, our algorithm can exploit the graph-theoretic structure of the problem and can thus find good solutions very quickly, even with provable quality guarantees.

Specific limitations of existing work include the following. Some approaches do not consider the costs of using the cloud [17], [32], or do not take data transfer between components into account [26], [28], [31], which, however, can lead to significant costs. In contrast, our algorithm explicitly minimizes the costs of both cloud resources and data transfer between the edge data center and the cloud.

Some approaches only work for a special application structure (e.g., directed acyclic graph [32], cycle of 4 vertices [33], series-parallel graph [17], [34]). In contrast, FOGPART works for any application topology.

Some approaches were not designed for dynamic runtime offloading. In several cases, algorithm performance is not evaluated [31], [36]. Some algorithms are too slow to be applied in a dynamic setting [26].

Some approaches used greedy algorithms [28], [33] that consider one application at a time and deploy its components sequentially. In contrast, our algorithm optimizes the deployment of all applications together, increasing the likelihood of finding overall good solutions, and uses special techniques to escape local optima. The experimental results clearly show that FOGPART outperforms FF, a typical example of a greedy sequential heuristic.

None of the approaches considers hard constraints on data protection in the offloading decisions. In contrast, we ensure that such data protection constraints are met.

9 Conclusions

This paper addressed the problem of deciding which application components to place in an edge data center and which ones in the cloud, subject to data protection, latency, and capacity constraints, minimizing the costs of cloud usage and data transfer between the edge data center and the cloud. We devised a new algorithm for this problem and proved that it delivers a deployment that meets all requirements whenever possible. In terms of costs, the deployment found by our algorithm is generally not optimal, but our experimental results suggest that it is close to optimal, while the algorithm is fast so that it can be used online. We also showed that the algorithm can be extended to effectively handle multiple edge data centers and end-to-end application latency optimization.

As future research, we aim to extend our approach to further optimization objectives and constraints, like energy consumption. An important question is to what extent the theorems of this paper can be transferred to other metrics.

Acknowledgments. Research leading to these results received funding from the EU's Horizon 2020 research and innovation programme under grant agreement no. 871525 (FogProtect).

References

[1] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the Internet of Things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[2] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything*. Springer, 2018, pp. 103–130.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 2017.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

[5] Z. Abbas, J. Li, N. Yadav, and I. Tariq, "Computational task offloading in mobile edge computing using learning automata," in *IEEE/CIC Int. Conf. on Communications in China*, 2018, pp. 57–61.

[6] Y. Nan, W. Li, W. Bao, F. C. Delicato, P. F. Pires, and A. Y. Zomaya, "A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems," *J. Parallel Distrib. Comput.*, vol. 112, pp. 53–66, 2018.

[7] M. Aazam, S. Zeadally, and K. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Gener. Comput. Syst.*, vol. 87, pp. 278–289, 2018.

[8] D. Bermbach, F. Pallas, D. G. Pérez, P. Plebani, M. Anderson, R. Kat, and S. Tai, "A research perspective on fog computing," in *Int. Conf. on Service-Oriented Computing*, 2017, pp. 198–210.

[9] S. Deng, Z. Xiang, J. Yin, J. Taheri, and A. Y. Zomaya, "Composition-driven IoT service provisioning in distributed edges," *IEEE Access*, vol. 6, pp. 54 258–54 269, 2018.

[10] V. B. C. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sánchez-López, J. Garcia, G. Ren, A. Jukan, and A. J. Ferrer, "Towards a proper service placement in combined fog-to-cloud (F2C) architectures," *Future Gener. Comput. Syst.*, vol. 87, pp. 1–15, 2018.

[11] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Int. Conf. on Service-Oriented Computing (ICSOC)*, 2018, pp. 230–245.

[12] P. Ravindra, A. Khochare, S. P. Reddy, S. Sharma, P. Varshney, and Y. Simmhan, "ECHO: An adaptive orchestration platform for hybrid dataflows across cloud and edge," in *Int. Conf. on Service-Oriented Computing (ICSOC)*, 2017, pp. 395–410.

[13] S. Schoenen, Z. Á. Mann, and A. Metzger, "Using risk patterns to identify violations of data protection policies in cloud systems," in *Service-Oriented Computing – ICSOC 2017 Workshops*. Springer, 2018, pp. 296–307.

[14] A. Brogi, S. Forti, C. Guerrero, and I. Lera, "How to place your apps in the fog: State of the art and open challenges," *Software: Practice and Experience*, vol. 50, no. 5, pp. 719–740, 2020.

[15] Z. Á. Mann, A. Metzger, J. Prade, and R. Seidl, "Optimized application deployment in the fog," in *Int. Conf. on Service-Oriented Computing (ICSOC)*. Springer, 2019, pp. 283–298.

[16] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of Experience (QoE)-aware placement of applications in fog computing environments," *J. Parallel Distrib. Comput.*, vol. 132, pp. 190–203, 2019.

[17] A. da Silva Veith, M. D. de Assuncao, and L. Lefevre, "Latency-aware placement of data stream analytics on edge computing," in *Int. Conf. on Service-Oriented Computing (ICSOC)*, 2018, pp. 215–229.

[18] Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, and F. Desprez, "Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed IoT applications in the fog," in *Proc. 33rd ACM Symposium on Applied Computing (SAC)*. ACM, 2018, pp. 751–760.

[19] C. H. Papadimitriou, *Computational Complexity*. John Wiley and Sons Ltd., 2003.

[20] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[21] M. Goudarzi, M. Palaniswami, and R. Buyya, "A distributed application placement and migration management techniques for edge and fog computing environments," in *16th Conference on Computer Science and Intelligence Systems (FedCSIS)*. IEEE, 2021, pp. 37–56.

[22] E. Ahvar, S. Ahvar, Z. Á. Mann, N. Crespi, R. Glitho, and J. Garcia-Alfaro, "DECA: a Dynamic Energy cost and Carbon emission-efficient Application placement method for edge clouds," *IEEE Access*, vol. 9, pp. 70 192–70 213, 2021.

[23] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Computing Surveys*, vol. 53, no. 3, p. art. 65, 2020.

[24] M. A. Al-Tarawneh, "Bi-objective optimization of application placement in fog computing environments," *Journal of Ambient Intelligence and Humanized Computing*, 2021.

[25] A. Brogi and S. Forti, "QoS-aware deployment of IoT applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[26] A. Brogi, S. Forti, and A. Ibrahim, "Predictive analysis to support fog application deployment," in *Fog and Edge Computing: Principles and Paradigms*. Wiley, 2019, pp. 191–222.

[27] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware fog service placement," in *IEEE 1st Int. Conf. on Fog and Edge Computing (ICFEC)*. IEEE, 2017, pp. 89–96.

[28] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in fog-cloud computing paradigm," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017, pp. 1222–1228.

[29] R. Deng, R. Lu, C. Lai, and T. Luan, "Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing," in *IEEE Int. Conf. on Communications*, 2015, pp. 3909–3914.

[30] V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," in *IEEE Int. Conf. on Communications (ICC)*, 2016, pp. 1–5.

[31] 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.

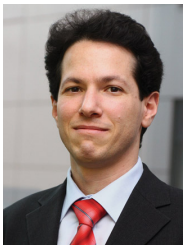
[32] X. Cai, H. Kuang, H. Hu, W. Song, and J. Lü, "Response time aware operator placement for complex event processing in edge computing," in *Int. Conf. on Service-Oriented Computing (ICSOC)*, 2018, pp. 264–278.

[33] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, p. 9, 2018.

[34] C. Mouradian, S. Kianpisheh, M. Abu-Lebdeh, F. Ebrahimnezhad, N. T. Jahromi, and R. Glitho, "Application component placement in NFV-based hybrid cloud/fog systems with mobile fog nodes," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1130–1143, 2019.

[35] C. Guerrero, I. Lera, and C. Juiz, "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures," *Future Gener. Comput. Syst.*, vol. 97, pp. 131–144, 2019.

[36] A. Gad-ElRab and A. Noaman, "A two-tier bipartite graph task allocation approach based on fuzzy clustering in cloud-fog environment," *Future Gener. Comput. Syst.*, vol. 103, pp. 79–90, 2020.



Zoltán Ádám Mann is senior researcher at paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany. His research interests include fog computing, software engineering, cloud computing, security and privacy, and optimization algorithms.

Andreas Metzger is senior academic councillor at the University of Duisburg-Essen and head of adaptive systems and big data applications at paluno, the Ruhr Institute for Software Technology. He is steering committee vice-chair of the European Technology Platform on Software, Services, Cloud and Data (NESSI) and deputy secretary general of the Big Data Value Association (BDVA). His research interests are in software engineering for self-adaptive systems.

Johannes Prade received his PhD in physics from the University of Regensburg, Germany. He has been active in the ICT field for more than 20 years. His research interest within Nokia is in investigating various aspects of network virtualization, including its organizational and monetary impact.

Robert Seidl is research manager at Nokia Bell Labs, heading a team responsible for Data Analytics and Privacy. He is also member of the Board of Directors at the Big Data Value Association (BDVA) and member of the Steering Committee of NESSI.

Klaus Pohl is full professor for Software Systems Engineering at the University of Duisburg-Essen. He is director of paluno – The Ruhr Institute for Software Technology and was scientific founding director of Lero – The Irish Software Engineering Centre. Klaus is member of the NESSI Board and member of several steering and advisory committees.

Cost-optimized, data-protection-aware offloading between an edge data center and the cloud – Supplementary material –

Zoltán Ádám Mann, Andreas Metzger, Johannes Prade, Robert Seidl, and Klaus Pohl

In this supplementary file, we describe the proofs of the theorems and propositions stated in the paper (Sec. 1), followed by the details of the baseline algorithms that FOGPART is compared with in the paper (Sec. 2) and a discussion of the limitations of and possible extensions to FOGPART (Sec. 3). Finally, some further experimental results are shown in Sec. 4 and an example for the coalescing of critical connectors in Sec. 5.

1 PROOFS

1.1 Solvability of the MCECD problem

We first recapitulate the basic equations and definitions from the paper.

A *valid* deployment must respect the following constraints:

$$\varrho(d) \leq P, \tag{1}$$

$$\forall v \in V: \ s(v) \Rightarrow (d(v) = \text{edge}), \tag{2}$$

$$\forall vw \in E: \ (\ell(vw) < L) \Rightarrow (d(v) = d(w)). \tag{3}$$

The cost of deployment d is defined as follows:

$$\text{cost}(d) = \sum_{v \in V,\ d(v)=\text{cloud}} c_{\text{p}} \cdot p(v) + \sum_{e \in E(d)} c_{\text{dt}} \cdot h(e). \tag{4}$$

Definition 1. *A connector $e \in E$ is* critical *if and only if $\ell(e) < L$.*

Remark 2. *According to Equation (3), if the connector vw is critical, then for any valid deployment, either both v and w must be in the edge data center or both of them must be in the cloud.*

Definition 3. *A component $v \in V$ is* critical *if and only if at least one of the following conditions hold:*

- *$s(v)$ is true; or*
- *There is a path $v_0, v_1, \ldots, v_k$ in the graph (V, E), such that $s(v_0) = true$, $v_k = v$ and for each $j = 1, \ldots, k$, the connector $v_{j-1}v_j$ is critical.*

In addition, each end device $v \in V_D$ is considered to be critical as well. Let $F = \{v \in V : v \text{ is critical}\}$ be the set of critical components and end devices.

Now we come to prove the following propositions.

Proposition 4. *Let $v \in V$ be critical. Then for any valid deployment d, $d(v) = edge$.*

Proof. If $s(v)$ is true (note that this also includes the case when $v \in V_{\text{D}}$), then Equation (2) ensures that $d(v) = \text{edge}$.

Otherwise, let $v_0, v_1, \ldots, v_k$ be a path as in Definition 3, and let d be a valid deployment. Then $d(v_0) = \text{edge}$ because of Equation (2). Since each connector $v_{j-1}v_j$ ($j = 1, \ldots, k$) is critical, it follows by induction and Remark 2 for each v_j ($j = 1, \ldots, k$) that $d(v_j) = \text{edge}$. For $j = k$, this completes the proof. □

Proposition 5. *A valid deployment exists if and only if*

$$\sum_{v \in F} p(v) \leq P. \tag{5}$$

If (5) holds, the following deployment is valid:

$$d(v) = \begin{cases} \textit{edge}, & \textit{if } v \in F; \quad (6a) \\ \textit{cloud}, & \textit{otherwise.} \quad (6b) \end{cases}$$

Proof. If deployment d is valid, then

$$\sum_{v \in F} p(v) \leq \sum_{v \in V,\ d(v)=\text{edge}} p(v) \leq P$$

(the first inequality follows from Proposition 4 and $p(v) \geq 0$, while the second inequality follows from (1)), which proves (5).

To see the reverse, assume that (5) holds, and let $d(v)$ be defined according to (6a)-(6b).

(6a) ensures that d satisfies (2). On the other hand,

$$\sum_{v \in V,\ d(v)=\text{edge}} p(v) = \sum_{v \in F} p(v) \leq P$$

(the equation follows from the definition of d, the inequality follows from (5)), which proves that d also satisfies (1).

To prove (3) by contradiction, let us assume that $vw \in E$ is a critical connector, $d(v) = \text{edge}$, and $d(w) = \text{cloud}$. Then, from the definition of d we know that $v \in F$. If $s(v) = \text{true}$, then this fact, together with the connector vw (which is a path of length 1) implies that $w \in F$, which contradicts $d(w) = \text{cloud}$. The other possibility is that there is a path $v_0, v_1, \ldots, v_k$ as in Definition 3, with $v_k = v$. However, this path, together with the connector vw implies that $w \in F$, which again contradicts $d(w) = \text{cloud}$. □

Algorithm 1 Adding a new application

```
1: procedure ADD(A)
2:     for v ∈ V_A do
3:         if s(v) then
4:             d(v) ← edge
5:         else
6:             d(v) ← cloud
7:         end if
8:     end for
9:     RE-OPTIMIZE(d)
10: end procedure
```

Algorithm 2 Removing an application

```
1: procedure REMOVE(A)
2:     for v ∈ V_A do
3:         remove v
4:     end for
5:     RE-OPTIMIZE(d)
6: end procedure
```

Algorithm 3 Handling changes

```
1: procedure CHANGES
2:     for v ∈ V do
3:         if s(v) and d(v) = cloud then
4:             d(v) ← edge
5:         end if
6:     end for
7:     RE-OPTIMIZE(d)
8: end procedure
```

1.2 Complexity of the MCECD problem

Next, we prove the strong NP-hardness of the defined optimization problem.

Theorem 6. *The MCECD problem is strongly NP-hard.*

Proof. We show a reduction from the minimum bisection problem. In an instance of the minimum bisection problem, we are given an undirected graph $G = (V, E)$ where $n = |V|$ is even, and a number $0 \leq S \leq |E|$. It has to be decided whether V can be partitioned into two disjoint sets V_1, V_2 of equal size such that the number of edges connecting the two subsets is at most S.

From this, we create an instance of the MCECD problem as follows. $V_{\mathrm{D}} = \emptyset$, and there is a single application, given by the graph G. For each component $v \in V$, $p(v) = M$ and $s(v)$ is false. Here, M is a number greater than $|E|$. For each connector $e \in E$, $h(e) = \ell(e) = 1$. Further, $L = 0$, $P = M \cdot n/2$, and $c_{\mathrm{p}} = c_{\mathrm{dt}} = 1$.

If G admits a bisection (V_1, V_2) with at most S edges between the two subsets, then there is a corresponding deployment that maps the components in V_1 to the edge data center and the components in V_2 to the cloud. It can be easily checked that this is a valid deployment, and its cost is at most $M \cdot n/2 + S$.

On the other hand, suppose that there is a deployment d with cost at most $M \cdot n/2 + S$. Then in this deployment exactly $n/2$ components must be in the edge data center and $n/2$ in the cloud. This is because the capacity constraint ensures that the number of components in the edge data center cannot be more than $n/2$, and if there were less than $n/2$ components in the edge data center, then the cost would be at least $M \cdot (n/2+1) > M \cdot n/2 + |E| \geq M \cdot n/2 + S$, which would be a contradiction. As a consequence, the number of connectors between the edge data center and the clouds is at most S. Hence $V_1 = \{v \in V : d(v) = \text{edge}\}$ and $V_2 = V \setminus V_1$ form a bisection with at most S edges between them.

Thus we have shown that the answer to the minimum bisection problem instance is equivalent to whether the constructed instance of the MCECD problem admits a solution with cost at most $M \cdot n/2 + S$. Since the minimum bisection problem is strongly NP-hard and all numbers in our construction are polynomially bounded (using for instance $M = |E| + 1$), this completes the proof. □

1.3 Complexity of FOGPART

We start by recapitulating the algorithms from the paper. These are shown in Algorithms 1-5. In addition, we also recapitulate the definition of *gain*.

Definition 7. *Let d be a deployment and $v \in V$ a component. Let d' be the deployment obtained from d by moving v. That is, for a component $w \in V$,*

$$d'(w) = \begin{cases} d(w) & \text{if } w \neq v, \\ \textit{edge} & \text{if } w = v \text{ and } d(v) = \textit{cloud}, \\ \textit{cloud} & \text{if } w = v \text{ and } d(v) = \textit{edge}. \end{cases}$$

Then, given deployment d, the gain of moving v is defined as

$$\mathit{gain}(d, v) = \begin{cases} -\infty & \text{if } d \text{ is valid, } d' \text{ is invalid}, \\ \mathit{cost}(d) - \mathit{cost}(d') & \text{otherwise}. \end{cases}$$

Next, we move on to prove the results on time complexity.

Theorem 8. *The time complexity of Algorithm 4 is $O\big(|V| \cdot (|V| + |E|)\big)$.*

Proof. The time complexity of Algorithm 4 is dominated by the *while* loop (lines 6-27) and the nested *for* loop (lines 8-16). In each pass of the *while* loop, one component is removed from L ("best_gain" $> -\infty$, line 20) or the loop terminates ("best_gain" $= -\infty$, line 26); therefore, the *while* loop is repeated at most $|V|$ times. In the *for* loop, each component in L is considered; in the worst case, this means all components in V. For a component v, the gain calculation takes $O(1 + \delta(v))$ steps, where $\delta(v)$ is the number of connectors incident to v. The remaining steps in the body of the *for* loop take $O(1)$ steps. Hence, the total time spent in the *for* loop is at most $\sum_{v \in V} O(1 + \delta(v)) = O(|V| + |E|)$. Together with the already established fact that the *while* loop is repeated at most $|V|$ times, this completes the proof. □

Corollary 9. *The time complexity of Algorithms 1, 2, and 3 is also $O\big(|V| \cdot (|V| + |E|)\big)$.*

Proof. For Algorithms 1, 2, and 3, all the steps before the re-optimization together take $O(|V|)$ steps. Thus the time complexity of these algorithms is dominated by the re-optimization. □

We end this subsection by proving the results on space complexity.

Algorithm 4 Deployment re-optimization

```
1: procedure RE-OPTIMIZE(d)
2:     best_deployment ← d
3:     best_cost ← cost(d)
4:     L ← {v ∈ V : ¬s(v)}
5:     end ← (L = ∅)
6:     while ¬end do
7:         best_gain ← −∞
8:         for v ∈ L do
9:             if ϱ(d) ≤ P or d(v) = edge then
10:                g ← GAIN(d,v)
11:                if g > best_gain then
12:                    best_comp ← v
13:                    best_gain ← g
14:                end if
15:            end if
16:        end for
17:        if best_gain > −∞ then
18:            forced ← (ϱ(d) > P)
19:            change d(best_comp) to the other value
20:            L.remove(best_comp)
21:            if forced or cost(d) < best_cost then
22:                best_deployment ← d
23:                best_cost ← cost(d)
24:            end if
25:        end if
26:        end ← (L = ∅ or best_gain = −∞)
27:    end while
28:    d ← best_deployment
29: end procedure
```

Algorithm 5 Calculation of the gain of moving a component

```
1: procedure GAIN(d, v)
2:     if d(v) = edge then
3:         r ← −c_p · p(v)
4:     else if ϱ(d) ≤ P and ϱ(d) + p(v) > P then
5:         return −∞
6:     else
7:         r ← c_p · p(v)
8:     end if
9:     for vw ∈ E do
10:        if d(v) = d(w) then
11:            r ← r − c_dt · h(vw)
12:        else
13:            r ← r + c_dt · h(vw)
14:        end if
15:    end for
16:    return r
17: end procedure
```

Theorem 10. *Algorithm 5 requires $O(1)$ auxiliary space. Algorithms 1–4 require $O(|V|)$ auxiliary space.*

Proof. In Algorithm 5, all partial results are accumulated in the variable r, thus requiring only $O(1)$ auxiliary space.

In Algorithm 4, the used auxiliary space is as follows:

- d, best_deployment, L: $O(|V|)$
- best_cost, end, best_gain, v, $\varrho(d)$, g, best_comp, forced, cost(d): $O(1)$

Therefore, Algorithm 4 uses $O(|V|)$ auxiliary space altogether.

Algorithms 1–3 only add one local variable (v), requiring $O(|1|)$ auxiliary space in addition to that required by Algorithm 4.

Finally, it is important to note that the depth of the call stack is $O(|1|)$ (Algorithms 1–3 call Algorithm 4, which in turn calls Algorithm 5). □

1.4 Correctness of FOGPART

We start by recapitulating the definition of a call sequence.

Definition 11. *A* call sequence *is a list $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_k)$, where each $\gamma_i \in \{add, remove, change\}$, depending on whether the ith call was to Algorithm 1 (adding an application), Algorithm 2 (removing an application), or Algorithm 3 (other change). The set of applications and the deployment after i calls are denoted by $\mathcal{A}^{(i)}$ and $d^{(i)}$, respectively. In particular, $\mathcal{A}^{(0)}$ and $d^{(0)}$ denote the set of applications respectively the deployment before the first call.*

Now we can prove our main theoretical result.

Theorem 12. *Starting from $\mathcal{A}^{(0)} = \emptyset$ and performing an arbitrary sequence of calls $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_k)$, if condition (5) is satisfied throughout (i.e., the deployment problem is solvable), then each call results in a valid deployment.*

Proof. To prove the validity of the resulting deployments, the satisfaction of conditions (1) and (2) must be proven (as explained in Section 3.5 of the paper, condition (3) is then also guaranteed). We will show both of them using induction according to k. For $k = 0$, when $\mathcal{A} = \emptyset$, it is clear that both conditions are satisfied.

Now suppose that after a sequence of $k - 1$ calls, both (1) and (2) are satisfied and we make a further call to Algorithm 1, Algorithm 2, or Algorithm 3. We first show that (2) is satisfied after the call as well. If $\gamma_k =$ remove, then first some components are removed which cannot lead to a violation of (2). If $\gamma_k =$ add, then first some components are added in such a way that ensures that (2) is not violated (see lines 3-4 of Algorithm 1). If $\gamma_k =$ change, then lines 3-5 of Algorithm 3 ensure that (2) is satisfied. Hence in all cases, when Algorithm 4 is called, (2) holds. During the run of Algorithm 4, only the deployment of components in L is changed (cf. lines 19, 12, and 8). Critical components are excluded from L (cf. line 4). Hence, all critical components remain in the edge data center, ensuring that (2) remains satisfied.

Next, we turn to (1). If $\gamma_k =$ remove, then first some components are removed which cannot lead to a violation of (1). However, if $\gamma_k =$ add, then first some components are added, which may invalidate (1). Also if $\gamma_k =$ change, the changes may invalidate (1). In all cases, Algorithm 4 is called in the end, so it suffices to show that, whether or not the starting deployment satisfies (1), the deployment resulting from Algorithm 4 satisfies it. This will be shown through proving the following three claims:

- As long as (1) is not satisfied, each iteration of the *while* loop of Algorithm 4 results in moving a component from the edge data center to the cloud.
- After some iterations of the *while* loop, (1) will be satisfied.
- If after j iterations of the *while* loop, (1) is satisfied, then it is satisfied also after j' iterations of the *while* loop, for any $j' > j$.

To prove the first claim, it should be noted that (1) not being satisfied means that $\varrho(d) > P$, which means that

the condition in line 9 of Algorithm 4 is satisfied only if $d(v) =$ edge. As a consequence, only components currently in the edge data center are considered in lines 9-15, hence at the end of the *for* loop, "best_comp" will be a component currently in the edge data center. This component is then moved in line 19 to the cloud.

To prove the second claim, it should be noted that the *while* loop is not abandoned prematurely. Lines 5, 6, 26, and 27 show that the loop is ended if $L = \emptyset$ or "best_gain" $= -\infty$. As long as (1) is not satisfied, the gain of no component will be $-\infty$, since only the gains of moves that would violate a valid deployment would be $-\infty$ (cf. lines 4-5 in Algorithm 5). Hence, as long as $L \neq \emptyset$ and (1) is not satisfied, the *for* loop of Algorithm 4 will investigate at least one component, its gain will be greater than $-\infty$, ensuring that at the end of the *for* loop "best_gain" $\neq -\infty$. Hence, the algorithm will move components from the edge data center to the cloud as long as $L \neq \emptyset$ and (1) is not satisfied. Should L get empty before (1) gets satisfied, that would mean that all non-critical components have already been moved to the cloud, i.e., only the critical components are in the edge data center, and (1) is still not satisfied. This, however, would contradict condition (5). Therefore, (1) gets satisfied after some iterations of the *while* loop.

To prove the third claim, it should be noted that once (1) is satisfied, any move that would violate (1) has gain $-\infty$ (cf. lines 4-5 of Algorithm 5). Such moves are not executed (cf. line 17 of Algorithm 4). Hence, (1) remains satisfied.

Putting the pieces together, we have established that after some iterations of the *while* loop, (1) is satisfied, and remains satisfied in the further iterations as well. Finally we need to show that the deployment that Algorithm 4 returns at the end as $d^{(k)}$ also satisfies (1). If the deployment already satisfies (1) at the beginning, then this is clear, since "best_deployment" is initialized with this deployment and later may only be overwritten by other deployments satisfying (1). Otherwise, as long as the deployment does not satisfy (1), lines 18 and 21 of Algorithm 4 ensure that the deployment after the current move is recorded in "best_deployment". This happens the last time when the deployment changes from invalid to valid (this is because the validity check of line 18 happens before the move is made). This ensures that a valid deployment gets stored in "best_deployment". In later iterations of the *while* loop, "best_deployment" may only be overwritten by other valid deployments; hence, when "best_deployment" is activated in line 28, it is guaranteed to be valid. □

2 BASELINE ALGORITHMS

In this section, we give more details about the two baseline algorithm to which FOGPART is compared in the paper.

2.1 Exact algorithm using integer linear programming

The operation of the ILP-based algorithm is shown in Algorithm 6. The ILP-based algorithm uses the Gurobi Optimizer[1], version 7.0.2, as an external solver to solve the ILP defined above. The algorithm uses the API provided by Gurobi for creating the variables (line 2) and the integer program (line 3), as well as to solve the integer program (line 4), to check if a solution could be found (line 5), and if this is the case, to query the value of the x_v variables in the found solution of the integer program to determine the corresponding solution to the original MCECD problem (line 6). In the following experiments, the ILP solver was executed in single-threaded mode with a timeout of 60 seconds. It can happen in two cases that the solver does not find a solution: either if there is no solution at all (cf. Proposition 5), or if a timeout prohibited the solver to find a solution. In both cases, the algorithm returns the information that no solution was found (line 9).

1. https://www.gurobi.com/

Algorithm 6 ILP-based algorithm

```
1:  procedure ILP
2:      Create variables {x_v : v ∈ V} and {y_e : e ∈ E}
3:      Create the integer program
4:      Invoke external ILP solver to solve the integer program
5:      if solution found then
6:          Determine solution from value of x_v variables
7:          return solution
8:      else
9:          return "no solution found"
10:     end if
11: end procedure
```

Algorithm 7 First-Fit algorithm

```
1:  procedure FF
2:      free ← P
3:      for v ∈ V do
4:          if s(v) then
5:              d(v) ← edge
6:              free ← free − p(v)
7:          end if
8:      end for
9:      if free < 0 then
10:         return "no solution found"
11:     end if
12:     for v ∈ V do
13:         if not s(v) then
14:             if p(v) ≤ free then
15:                 d(v) ← edge
16:                 free ← free − p(v)
17:             else
18:                 d(v) ← cloud
19:             end if
20:         end if
21:     end for
22:     return d
23: end procedure
```

2.2 First-fit heuristic

The First-Fit (FF) heuristic is shown in Algorithm 7. It first places all critical components into the edge data center, while keeping track of the free capacity of the edge data center (lines 2-8). If the critical components require more capacity than what is available in the edge data center, then the algorithm returns the information that no solution could be found (lines 9-11). Otherwise, the algorithm iterates over the remaining components, and does the following for each non-critical component v: if v still fits into the edge data center, then v is placed into the edge data center (lines

14-16), otherwise into the cloud (lines 17-19). Finally, the algorithm returns the deployment created this way (line 22).

3 Discussion

In this section, we discuss some further details and limitations of the proposed problem model and algorithm, as well as possible extensions.

3.1 Migrations

When re-optimizing the deployment of existing software components, the overhead caused by migrating the components may be a concern. Although modern virtualization technologies provide acceptable overhead, if many components have to be migrated at the same time, this may still be a problem [1].

Also the cost aspects of migrations may matter. Since migrations also involve data transfer, they also incur costs, which is not captured by the current problem model. The rationale for not including the costs caused by migrations in the problem model is that migration costs are incurred only once, whereas the costs considered in the problem formulation are running costs, which are incurred continuously. As long as migrations are not frequent, the cost impact of migrations can be neglected when compared to the running costs.

For the above reasons, it is disadvantageous if a re-optimization algorithm leads to a high number of migrations. During the experiments reported in the paper, the average number of migrations per run was 14.7 for ILP, 10.3 for FogPart, and 8.7 for FF. Together with the other experimental results, this means that FogPart achieves nearly as low costs as ILP, but with significantly less migrations. When compared to FF, FogPart makes more migrations, but this also results in considerably better results. Therefore, FogPart achieves a good trade-off between migrations and solution costs.

On the other hand, if the number of migrations is critical (this depends on many factors, such as the used virtualization technology and the available bandwidth), FogPart can be easily extended to limit the number of migrations. Since FogPart moves components one after the other, the algorithm can simply be terminated if the allowed number of migrations has been exhausted. Limiting the number of migrations in the ILP and FF algorithms would require more extensive modifications.

3.2 Costs and constraints of the edge data center

In the current problem formulation, it is assumed that using the resources of the edge data center is free, since they are at the disposal of the provider of the edge data center. A possible extension would be to assume that deploying component v in the edge data center incurs a cost of $c_0 \cdot p(v)$, where c_0 is a given constant. This extension can be reduced to the special case handled so far with the following transformation:

$$\sum_{v \in V,\ d(v)=\text{edge}} c_0 \cdot p(v) + \sum_{v \in V,\ d(v)=\text{cloud}} c_\text{p} \cdot p(v) = \\ = \sum_{v \in V} c_0 \cdot p(v) + \sum_{v \in V,\ d(v)=\text{cloud}} (c_\text{p} - c_0) \cdot p(v), \quad (7)$$

where the first term is a constant and the second term is the same as in the original problem formulation with $c'_\text{p} = c_\text{p} - c_0$.

Currently, our problem formulation only focuses on CPU capacity. However, it is straight-forward to extend it with further constraints, for instance with respect to memory capacity.

3.3 Optimization objective

Our current problem formulation and algorithms handle the optimization of the sum of cloud usage costs and data transfer costs. The objective function could be easily extended with further quality metrics, like energy consumption or reliability. For FogPart to work, it only has to be ensured that the gain of moving a component from the edge data center to the cloud or vice versa can be efficiently computed. This property holds as long as the objective function is a weighted sum of quality metrics, in which each vertex has an identifiable additive contribution.

If the contribution of individual vertices to a considered quality metric is multiplicative (as is the case with availability for example), then the logarithm of the quality metric could be used to ensure additivity, as done by Nardelli et al. [2].

3.4 Cloud pricing schemes

Our problem formulation and algorithms assume a single unit price for compute power in the cloud. This abstracts from the many different prices that a cloud provider may actually offer. On the one hand, different virtual machine types may be associated with different ratios between price and computing power. On the other hand, a cloud provider may offer different pricing schemes based on the provider's and the user's commitment, such as discounts for reserved instances or for spot instances. In our scenario, the provider of the edge data center will probably use only a small subset of the possible instance types and pricing schemes offered by the cloud provider for allowing the offloading of excess components (for example, only small on-demand instances). Hence using a single unit price is a good approximation. Nevertheless, existing work on cloud cost optimization may be exploited to re-arrange the workload that was determined by our algorithm to run in the cloud with the aim of reducing costs [3], [4]. In addition, when dealing with reserved instances or spot instances, also temporal aspects become important since such offerings are for given periods of time. Hence, existing research on time-aware cloud resource allocation may be exploited to minimize costs over a given period of time [5], [6].

3.5 Further generalizations

We assumed that using the cloud is associated with costs and with data protection risks. These are typical characteristics of public cloud services. Our approach can also be used with a private cloud. In this case, the costs could be set to zero ($c_\text{p} = 0$), and the data protection requirements of the components could be omitted ($\forall A \in \mathcal{A}, \forall v \in V_A : s(v) =$ false).

FOGPART might be extended to directly handle more than one edge data center and/or more than one cloud, using a generalization of the Kernighan-Lin algorithm for graph partitioning to handle multiway cuts [7]. However, this is a non-trivial extension, which we leave for future work.

Another possible extension would be to support different unit prices for data transfer from the edge data center to the cloud and vice versa. This could be used to model the policy of some cloud providers to charge different prices for data transfers into and out of their cloud. Such a generalization would require the usage of a directed graph instead of an undirected graph to model the connectors, so as to differentiate the direction of data transfer. Such an extension seems possible, but would require a number of changes in the problem model, the algorithm, and the theoretical analysis.

3.6 Parallelization opportunities

As shown in Section 4.3 of the paper, FOGPART can be extended to the decentralized management of multiple edge data centers. In such a setting, the different FOGPART instances may run in parallel.

It is also possible to parallelize FOGPART within a single edge data center. The key idea is that the computation of the gains of different components can be done in parallel. Supposing that sufficient parallel resources are available, it is clear from the proof of Theorem 8 that the time complexity of FOGPART can thus be reduced to $O(|V| \cdot \Delta)$ parallel time, where Δ is the maximum number of connectors incident to a component. Hence, if there is a sufficient number of devices with free capacity within the edge data center, the execution time of FOGPART can be significantly reduced even compared to the already quite fast sequential execution.

3.7 Worst-case performance

Here we discuss what bounds could be given on the worst-case deviation of the costs achieved by FOGPART from the optimum. One possible approach would be to derive a bound on the *approximation ratio* of FOGPART, i.e., the ratio of the costs achieved by FOGPART to the optimum. Recall from the proof of Theorem 6 that the MCECD problem essentially contains the Minimum Bisection problem as special case. Approximating the Minimum Bisection problem is challenging in itself; the best known approximation algorithm for Minimum Bisection has an approximation ratio of $O(\log n)$ [8]. Thus, no constant approximation ratio should be expected for FOGPART.

Indeed, the following construction shows that the cost achieved by FOGPART can be arbitrarily far from the optimum. Let $V_A = \{v_1, \ldots, v_n\}$, where $n > 3$. The vertices $v_2, \ldots, v_n$ form a complete graph, while v_1 is isolated. Let $p(v_1) = 2$ and $p(v_i) = 1$ for each $2 \leq i \leq n$. Let $s(v_i) =$ false for each $1 \leq i \leq n$. For each connector e, $h(e) = H$ for some given $H > 1$. Let $P = n - 1$, $L = 0$, $c_{\mathrm{p}} = c_{\mathrm{dt}} = 1$, and $V_{\mathrm{D}} = \emptyset$. One possible solution of this instance of the MCECD problem is to place components $v_2, \ldots, v_n$ in the edge data center, and component v_1 in the cloud. The cost of this solution is 2, thus the optimum is at most 2. On the other hand, FOGPART first tentatively places all components in the cloud, and then makes the move with the highest gain, which moves v_1 to the edge data center. Afterwards, FOGPART successively moves $n-3$ of the remaining components to the edge data center, until the capacity of the edge data center is exhausted. At this point, the algorithm terminates with the best partition encountered. This is either the partition after the first move, with a cost of $n - 1$, or a partition in which at least one connector crosses the boundary between edge data center and cloud, leading to a cost of at least H. Thus, the resulting cost is at least $\min(n - 1, H)$. Since both $n - 1$ and H can be arbitrarily large, the resulting cost can be arbitrarily far from the optimum.

The empirical results of the paper show that such extreme situations do not typically occur. In typical situations, the result achieved by FOGPART is quite near to the optimum.

A different approach to quantifying the worst-case performance of FOGPART would be to calculate its *competitive ratio*, i.e., how it performs relative to a fictitious algorithm that knows all future inputs. However, this is only applicable to online problems, and the MCECD problem is not an online problem. This is because the MCECD problem only considers the reoptimization of the placement at a given point in time, without any references to the past or the future. In particular, any placement decisions in the past can be changed during the reoptimization. Also the optimization objective refers only to the result of the reoptimization at the given point in time. Accordingly, FOGPART is not an online algorithm. It would be possible to change the MCECD problem to an online problem, for example, by limiting the number of changes that are possible and by minimizing the costs aggregated over a period of time, involving multiple reoptimizations. FOGPART could be changed to an online algorithm, addressing this modified version of the MCECD problem. Then, the competitive ratio of this modified algorithm could be investigated.

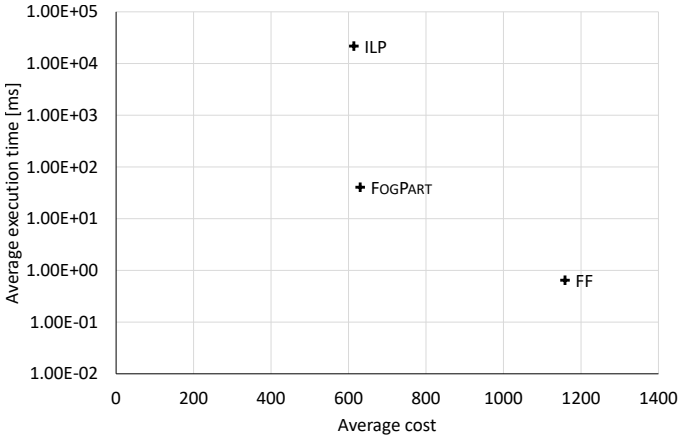


Fig. 1: Trade-offs between solution costs and execution time

4 ADDITIONAL EXPERIMENTAL RESULTS

4.1 Trade-offs of different algorithms

Summarizing the results of Sec. 7.2-7.5 of the paper, Fig. 1 shows the different trade-offs between solution costs and

TABLE 1: Execution time for multiple edge data centers

c_{dt}	$\frac{c_{\text{ee}}}{c_{\text{dt}}}$	Execution time [ms]			
		FF	ILP	DISTFOGPART	CROSSFOGPART
0.009	0.01	2	1188	12	423
0.009	0.1	2	1059	3	466
0.009	1	1	899	5	438
0.09	0.01	1	1034	2	331
0.09	0.1	0	960	4	335
0.09	1	2	745	4	367
0.9	0.01	2	397	0	390
0.9	0.1	3	263	4	348
0.9	1	2	344	5	390

execution time achieved by the algorithms. Qualitatively, it was awaited that the results of FOGPART are somewhere between the results of the other two algorithms, but the experimental results show this quantitatively. It can clearly be seen that FOGPART is almost as good as ILP in terms of solution costs, while being almost as fast as FF (note the logarithmic scale of the vertical axis). FOGPART is very fast and consistently leads to near-optimal results, thus representing a very attractive trade-off.

4.2 Execution time for multiple edge data centers

Sec. 7.6 of the paper shows results in terms of costs for multiple edge data centers. Here, we provide the corresponding execution times. Note that the execution times reported for this and the next experiment were achieved on a newer computer than the results in previous experiments. The new computer is a laptop with Intel i7-1165G7 CPU @ 2.80 GHz and 16 GB RAM, running Windows 10, Java HotSpot 64-bit VM version 16.0.2, and Gurobi Optimizer version 9.1.2.

The results are shown in Table 1. As in previous experiments, the execution time of FF is low and that of ILP orders of magnitude higher. In line with the results of previous experiments, the execution time of DISTFOGPART is somewhat higher than that of FF, but still orders of magnitude lower than that of ILP. However, the execution time of CROSSFOGPART is significantly higher that that of DISTFOGPART. This is because CROSSFOGPART performs additional optimization steps in all other edge data centers after an application was added to one edge data center. These additional optimization steps in the other edge data centers are currently done sequentially in our implementation. These steps could be done in parallel, leading to significantly lower execution time.

4.3 Execution time for global latency optimization

Sec. 7.7 of the paper shows results in terms of costs for end-to-end application latency optimization. Here, we provide the corresponding execution times in Fig. 2. As in previous experiments, the execution time of FF is low and that of ILP orders of magnitude higher, with that of GLOFOGPART in between. As λ increases, the execution time of ILP decreases, but it is always significantly higher than that of GLOFOGPART (note the logarithmic scale of the vertical axis).

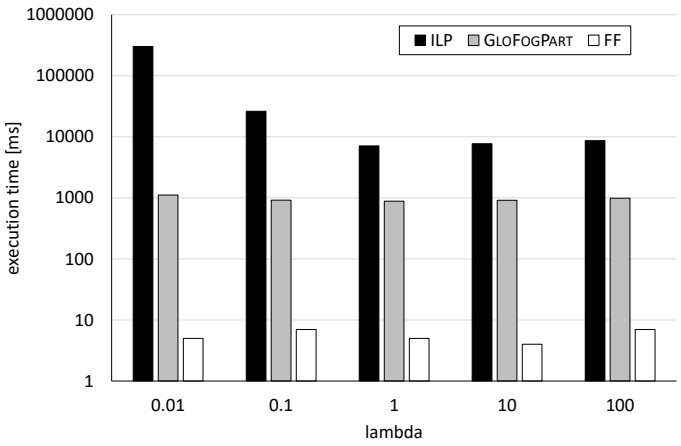


Fig. 2: Execution time for different values of λ

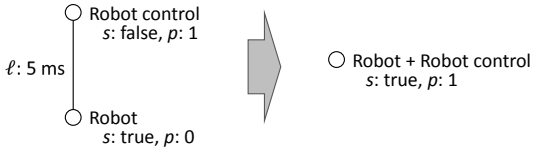


Fig. 3: Example for the coalescing of critical connectors

5 EXAMPLE FOR THE COALESCING OF CRITICAL CONNECTORS

In Sec. 3.4 of the paper, a transformation of the MCECD problem was introduced, which relies on the coalescing of critical connectors. Fig. 3 shows an example. The maximum allowed latency of the connector between the Robot control component and the Robot device is 5 ms, while the latency between the edge data center and the cloud is 100 ms, making this connector a critical one. After coalescing this connector, the two vertices are replaced by a single one, with $p(w) = p(u) + p(v) = 1 + 0 = 1$ and $s(w) = s(u) \vee s(v) = \text{false} \vee \text{true} = \text{true}$.

ACKNOWLEDGMENTS

Research leading to these results received funding from the EU's Horizon 2020 research and innovation programme under grant agreement no. 871525 (FogProtect).

REFERENCES

[1] D. Bartók and Z. Á. Mann, "A branch-and-bound approach to virtual machine placement," in *Proceedings of the 3rd HPI Cloud Symposium "Operating the Cloud"*, 2015, pp. 49–63.

[2] M. Nardelli, V. Cardellini, V. Grassi, and F. L. Presti, "Efficient operator placement for distributed data stream processing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1753–1767, 2019.

[3] S. Chaisiri, R. Kaewpuang, B.-S. Lee, and D. Niyato, "Cost minimization for provisioning virtual servers in amazon elastic compute cloud," in *IEEE 19th Annual Int. Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE, 2011, pp. 85–95.

[4] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth, "A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conf.*, 2013, pp. 1–10.

[5] R. B. Halima, S. Kallel, W. Gaaloul, and M. Jmaiel, "Scheduling business process activities for time-aware cloud resource allocation," in *OTM Confederated Int. Conf.s "On the Move to Meaningful Internet Systems"*. Springer, 2018, pp. 445–462.
[6] ——, "Optimal cost for time-aware cloud resource allocation in business process," in *IEEE Int. Conf. on Services Computing (SCC)*. IEEE, 2017, pp. 314–321.
[7] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, "Engineering a direct k-way hypergraph partitioning algorithm," in *Proceedings of the 19th Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2017, pp. 28–42.
[8] H. Räcke, "Optimal hierarchical decompositions for congestion minimization in networks," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, 2008, pp. 255–264.