Decentralized application placement in fog computing

Zoltán Ádám Mann

Abstract—In recent years, cloud computing concepts have been extended towards the network edge, leading to paradigms like fog and edge computing. As a result, applications can be placed on a variety of resources, including fog nodes and cloud data centers. Application placement has significant impact on important metrics like latency. Finding an optimal application placement is computationally challenging, particularly because of the potentially huge number of infrastructure nodes and application components. To overcome the limited scalability of application placement algorithms, optimization can be decentralized, i.e., performed separately for different parts of the infrastructure. The infrastructure can be split into fog colonies, where a fog colony consists of the computational resources in a given geographical region. Application placement can then be performed for the individual fog colonies, thus mitigating the scalability problem. However, independent optimization of application placement in different fog colonies may lead to missed synergies and thus to sub-optimal overall results. Hence, some kind of coordination between fog colonies may be beneficial.
In this paper, we analyze the effects of decentralization and coordination on the optimization results. In particular, we compare empirically four different approaches: (i) centralized decision-making, where decisions are made in one go for the entire infrastructure, (ii) independent fog colonies, where optimization is carried out in each fog colony independently from each other, (iii) fog colonies with communication, where excess application components in one fog colony can be sent to a neighboring fog colony, and (iv) fog colonies with overlaps, where shared resources may be dynamically distributed between neighboring fog colonies. Our experiments show that, for large problem instances, decentralization combined with coordination leads to the best results.

Index Terms—Fog computing, edge computing, application placement, distributed algorithms, fog colonies.

1 Introduction

A rapidly increasing number of connected smart devices in the Internet of Things (IoT) produce large amounts of data. Often, these end devices have limited computational resources or are constrained by their battery capacity [1]. As a result, data processing has to be – fully or partially – offloaded from the end devices. One possibility is to offload data processing to the cloud. However, the latency of transferring data between end devices and the cloud may be problematic for some applications [2]. To overcome this issue, computational resources called fog nodes can be deployed in a geographically distributed manner, so that end devices can offload data processing tasks to nearby fog nodes. This way, cloud-like services can be used with lower latency [3]. The cloud can still be used for tasks that are not latency-critical and that are especially resource-intensive.

The result (called edge computing or fog computing) is a distributed infrastructure with three layers, which respectively comprise the end devices, the fog nodes, and the cloud [4]. Software applications can be deployed to this distributed infrastructure. It has to be decided for each component of a software application, which cloud or fog node should host the given component. The decision of where to place the components of the application can have significant consequences on the performance of the application [5], [6]. For this reason, several algorithms have been proposed for finding a good placement of application components.

Most of the approaches proposed so far are based on centralized, global decision-making [7]. That is, information about the whole infrastructure and all application components is collected by a central entity, which then makes a decision based on all this information. Unfortunately, finding the best placement of application components is a computationally challenging problem. Hence, the scalability of the proposed algorithms is a major concern. Many of the algorithms proposed in the literature were only tested on small problem instances (just a handful of fog nodes and application components), and it is questionable how they would perform on larger problem instances [8].

One way to handle scalability issues is to abstain from centralized, global decision-making. A natural way to make fog application placement more efficient is to limit the scope of the problem to just one region. That is, only information about the infrastructure in a given region, along with the application components to be placed on that infrastructure, is considered. A good placement is then determined for the given region. This way, as long as the size of the regions is not too big, placement decisions can be made efficiently, even if the total size of the infrastructure and of the applications is large. The notion of *fog colonies* was introduced to describe the fog infrastructure in a given region, which hosts a limited set of application components [9].

Decentralized fog application placement in independent fog colonies improves scalability. However, an intrinsic limitation of such an approach is that, in general, it cannot achieve a global optimum. For instance, it is possible that

This paper was published in IEEE Transactions on Parallel and Distributed Systems, *33(12):3262-3273, 2022, https://doi.org/10.1109/TPDS.2022.3148985*

- *Z. Á. Mann is with the University of Amsterdam, Amsterdam, The Netherlands*

one fog colony is overloaded, while there is free capacity in a neighboring fog colony. Using centralized, global decision-making, load could be balanced between the two regions, leading to good overall results. In contrast, if each fog colony is optimized independently, such load balancing between regions is not possible. Therefore, some coordination between fog colonies may need to be introduced to achieve such load balancing.

In this paper, we investigate the impact of different decentralization and coordination schemes on efficiency and effectiveness. One extreme is the centralized approach, which can lead to the best overall results (high effectiveness), but suffers from poor scalability (low efficiency). The other extreme is the independent handling of fog colonies, which offers high efficiency but low effectiveness. In addition to these two extremes, we investigate two sophisticated coordination models. In the "fog colonies with communication" model, excess components in a fog colony can be transferred to a neighboring fog colony. In the "fog colonies with overlap" model, fog nodes may belong to two fog colonies, in which case their capacity can be dynamically distributed between the two involved fog colonies.

These decentralization and coordination models were proposed in the literature for different formulations of the fog application placement problem and evaluated with different placement algorithms, in different environments. Without a systematic comparison, it is not clear how these decentralization and coordination models compare to each other quantitatively, and which one will give the best result in which circumstances. To address this gap, this paper makes the following contributions:

- We formulate the above four decentralization and coordination schemes in a common formal framework.
- We implement the four investigated decentralization and coordination schemes in a consistent way.
- We implement an exact and a heuristic placement algorithm that can be used with all four investigated decentralization and coordination schemes.
- We compare empirically the four investigated decentralization and coordination schemes in terms of efficiency and effectiveness.
- We perform controlled experiments to determine the impact of different problem characteristics on the efficiency and effectiveness of the four investigated decentralization and coordination schemes.

To our knowledge, this is the first systematic study of the efficiency and effectiveness of different decentralization and coordination schemes in fog application placement.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 introduces and formalizes the addressed problem. Section 4 describes the investigated solution approaches. The results of the experiments are presented in Section 5. Section 6 discusses possible extensions. Finally, Section 7 concludes the paper.

2 Related work

Recent surveys discuss the variants of the fog application placement problem considered in the literature and the proposed solution approaches [7], [10]. It also becomes clear from these surveys that the majority of the papers propose centralized approaches. In the following, we review work on decentralized fog application placement.

Singh et al. address latency- and privacy-aware placement of applications in the fog [11]. They propose a distributed orchestration architecture based on agents on each fog and cloud node. Their algorithm is based on a few simple heuristic rules. The algorithm was compared only to a trivial cloud-only approach, using small problem instances.

The concept of fog colonies was first introduced in [12]. In that paper, Skarlat et al. introduced a fog architecture based on fog colonies. Their optimization approach only considered one fog colony, and forwarded requests that the colony could not host to the cloud. This was always possible because no latency constraints were considered. In [13], Skarlat et al. extended that work, such that requests that cannot be accommodated by a given fog colony may be sent both to the cloud and to the closest neighbor colony. The resulting optimization problem for a fog colony was solved using integer linear programming. Each fog colony is managed by a dedicated control node, and coordination among fog colonies is realized by communication among their control nodes. In a third paper, Skarlat et al. extended this work with further details and a genetic algorithm [9].

Other related work also adopted the concept of fog colonies as a basis for their placement approaches. Minh et al. proposed a service placement mechanism with the aim of maximizing the number of tasks deployed to the fog landscape [14]. Their approach is focused on optimization within a fog colony, with the possibility of forwarding tasks to a neighboring fog colony or to the cloud.

A similar concept was introduced by Alsaffar et al. [15]. They assume a cloud environment and a set of fog environments, where each environment offers some virtual machines. They propose a set of rules to determine if the received application services can be allocated locally or should be delegated to other environments, under the assumption that information about the current status of each environment is available in each environment. The effects of the network structure are not taken into account.

Also Shurman and Aljarah consider a similar setup [16]. They use different terminology, but essentially, they propose an algorithm that first tries to place components in the given fog colony, and then tries to send excess components to neighboring fog colonies. Components that cannot be allocated in any fog colony are forwarded to the cloud.

Baresi et al. proposed an approach using communities instead of fog colonies [17]. The main difference is that fog colonies are disjoint, whereas communities can overlap. Using communities requires inter-community allocation, i.e., splitting the resources of fog nodes belonging to multiple communities between those communities. Baresi et al. propose to base this decision on the aggregate demand and capacity of the involved communities.

Tong et al. propose to organize fog nodes in a rooted tree, where computation requests arrive at the leaves and can be either fulfilled in the given leaf or forwarded towards the root [18]. A branch-and-bound algorithm is suggested to allocate requests among the fog nodes of the path from leaf to root. In addition, a simulated annealing algorithm is

TABLE 1: Inputs

Symbol	Meaning
V_{SW}	Set of application components
c_v	CPU requirement of component $v \in V_{\text{SW}}$
r_v	RAM requirement of component $v \in V_{\text{SW}}$
$\mathcal{E}$	Set of end devices
E_{SW}	Set of connectors
b_e	Bandwidth requirement of connector $e \in E_{\text{SW}}$
d_e	Maximum allowed latency of connector $e \in E_{\text{SW}}$
V_{HW}	Set of infrastructure nodes (fog and cloud)
C_n	CPU capacity of infrastructure node $n \in V_{\text{HW}}$
R_n	RAM capacity of infrastructure node $n \in V_{\text{HW}}$
E_{HW}	Set of infrastructure links
B_l	Bandwidth of link $l \in E_{\text{HW}}$
D_l	Latency of link $l \in E_{\text{HW}}$
V^0_{SW}	Set of application components that are already placed
h_v	Infrastructure node that hosts component $v \in V^0_{\text{SW}}$
$\overline{V_{\text{SW}}}$	Set of application components and end devices
$\overline{V_{\text{HW}}}$	Set of infrastructure nodes and end devices

TABLE 2: Results of preprocessing the input

Symbol	Meaning
$P_{n,n'}$	Set of paths between vertices $n, n' \in \overline{V_{\text{HW}}}$
P	Set of all paths between all pairs of vertices in $\overline{V_{\text{HW}}}$
$D(p)$	Latency of path $p \in P$
$P(l)$	Set of all paths passing through link $l \in E_{\text{HW}}$

TABLE 3: Outputs to be computed by optimization

Symbol	Meaning
α	Placement of components on infrastructure nodes
β	Routing of connectors through infrastructure paths

proposed to allocate the capacity of a non-leaf node between the paths that the node is part of.

Some papers take the concept of decentralization to the extreme, so that each fog node makes decisions on its own, coordinating with neighboring nodes in a peer-to-peer manner. Lee et al. propose an approach by which a fog node can identify an appropriate subset of its neighboring nodes as offloading targets [19], [20]. Yousefpour et al. define policies by which fog nodes can decide if they process a request locally, forward it to the best neighboring node, or forward it to the cloud [21], [22]. Guerrero et al. propose an algorithm for fog nodes to decide if they can host a new service request or forward it to the next node on the path to the cloud [23]. Aral and Ovatman propose a decentralized algorithm for data replica placement, which is easier than application placement, since the communication among components does not have to be taken into account [24].

We can conclude that existing decentralized approaches address different variants of the fog application placement problem. The approaches proposed so far were mainly compared to trivial alternatives, like placing everything in the cloud. Thus, it is not clear how existing approaches compare to each other or for what problem variants and parameter settings they are appropriate. Our work fills this gap by implementing different decentralization and coordination approaches in a consistent way and systematically comparing them by means of controlled experiments.

3 Problem Model

In this section, we define the addressed version of the fog application placement problem, first for the whole infrastructure (Section 3.1). We show in Section 3.2 that this problem is strongly NP-hard. We refine the problem formulation in Section 3.3 to support fog colonies.

3.1 Basic problem model

Applications and end devices. We are given a set of applications. Each application consists of components. Also a set of end devices is given. A connector may connect either a pair of components, or a component and an end device. The set of all components in all applications is denoted by V_{SW} (see also Table 1). Each component $v \in V_{\text{SW}}$ has given CPU requirements $c_v \in \mathbb{R}_{\geq 0}$ and RAM requirements $r_v \in \mathbb{R}_{\geq 0}$. The set of end devices is denoted by $\mathcal{E}$. The set of all connectors is denoted by E_{SW}, where $E_{\text{SW}} \subseteq (V_{\text{SW}} \times V_{\text{SW}}) \cup (V_{\text{SW}} \times \mathcal{E})$. Each connector $e \in E_{\text{SW}}$ is associated with given bandwidth requirements $b_e \in \mathbb{R}_{\geq 0}$ and a maximum allowed latency $d_e \in \mathbb{R}_{\geq 0}$.

Infrastructure. The infrastructure is given by a set of nodes V_{HW} and a set of links E_{HW}. A link may connect either two nodes or a node and an end device, i.e., $E_{\text{HW}} \subseteq (V_{\text{HW}} \times V_{\text{HW}}) \cup (V_{\text{HW}} \times \mathcal{E})$. Each node $n \in V_{\text{HW}}$ has given CPU capacity $C_n \in \mathbb{R}_{\geq 0}$ and RAM capacity $R_n \in \mathbb{R}_{\geq 0}$. Each link $l \in E_{\text{HW}}$ has given bandwidth $B_l \in \mathbb{R}_{\geq 0}$ and latency $D_l \in \mathbb{R}_{\geq 0}$.

Let $\overline{V_{\text{HW}}} = V_{\text{HW}} \cup \mathcal{E}$. Based on the graph structure $(\overline{V_{\text{HW}}}, E_{\text{HW}})$, we compute a set of paths between each pair of vertices in $\overline{V_{\text{HW}}}$. This is a preprocessing step that only has to be done once. The set of computed paths between vertices $n, n' \in \overline{V_{\text{HW}}}$ is denoted by $P_{n,n'}$ (see also Table 2). The set of all computed paths is $P = \cup\{P_{n,n'} : n, n' \in \overline{V_{\text{HW}}}\}$. For a path $p \in P$, the latency of the path is the sum of the latencies of the links in the path: $D(p) = \sum_{l \in p} D_l$. For a link $l \in E_{\text{HW}}$, the set of paths passing through l is $P(l) = \{p \in P : l \in p\}$.

Placement. Our aim is to determine the functions $\alpha : V_{\text{SW}} \to V_{\text{HW}}$ and $\beta : E_{\text{SW}} \to P$ (cf. Table 3). For a component $v \in V_{\text{SW}}$, $\alpha(v) \in V_{\text{HW}}$ is the node that should host v. For a connector $e \in E_{\text{SW}}$, $\beta(e) \in P$ is the path through which e should be routed. The notation $\alpha^{-1}(n)$ is used to denote the set of components that node $n \in V_{\text{HW}}$ should host.

Over time, new applications can be deployed or existing applications can be removed. At a general point in time, some components may already be placed, while others are yet to be placed. We denote by $V^0_{\text{SW}} \subseteq V_{\text{SW}}$ the set of components that are already placed. For a component $v \in V^0_{\text{SW}}$, $h_v \in V_{\text{HW}}$ denotes the node currently hosting v.

Constraints. For a uniform handling of constraints, we define $\overline{V_{\text{SW}}} = V_{\text{SW}} \cup \mathcal{E}$. We extend α to a function $\overline{V_{\text{SW}}} \to \overline{V_{\text{HW}}}$, with the following rules:

$$v \in V_{\text{SW}} \;\Rightarrow\; \alpha(v) \in V_{\text{HW}}, \tag{1}$$

$$v \in \mathcal{E} \;\Rightarrow\; \alpha(v) = v. \tag{2}$$

The functions α and β must be consistent, i.e., a connector between vertices v_1 and v_2 in $\overline{V_{\text{SW}}}$ must be routed through a path between the corresponding nodes in $\overline{V_{\text{HW}}}$:

$$e = v_1 v_2 \in E_{\text{SW}} \;\Rightarrow\; \beta(e) \in P_{\alpha(v_1),\alpha(v_2)}. \tag{3}$$

TABLE 4: Notation related to fog colonies

Symbol	Meaning
K	Set of all fog colonies
$K(n)$	Set of fog colonies containing infrastructure node n
k_n	The single fog colony containing fog node n
k_v	The single fog colony assigned to component v

The CPU and RAM capacity of nodes must not be exceeded by the components they host:

$$\forall n \in V_{\text{HW}} : \sum_{v \in \alpha^{-1}(n)} c_v \leq C_n, \tag{4}$$

$$\forall n \in V_{\text{HW}} : \sum_{v \in \alpha^{-1}(n)} r_v \leq R_n. \tag{5}$$

The bandwidth of a link must not be exceeded by the bandwidth requirements of the connectors routed through that link:

$$\forall l \in E_{\text{HW}} : \sum_{e \in E_{\text{SW}},\ \beta(e) \in P(l)} b_e \leq B_l. \tag{6}$$

Each connector e must be routed through a path whose latency does not exceed the maximum allowed latency of e:

$$\forall e \in E_{\text{SW}} :\ D(\beta(e)) \leq d_e. \tag{7}$$

Objective. Our main objective is to find functions α and β that satisfy constraints (1)-(7). If there are multiple such solutions, we prefer the ones that lead to few migrations. The number of migrations is

$$M = \left|\{v \in V_{\text{SW}}^0 : \alpha(v) \neq h_v\}\right|. \tag{8}$$

3.2 Complexity

It can be easily seen that the defined problem includes the well-known bin packing problem as a special case. Namely, let us assume that there are no links, no connectors, no end devices, the RAM requirement of each component is 0, and $V_{\text{SW}}^0 = \emptyset$. Thus, the only constraint is the CPU constraint of the nodes (equation (4)). In addition, the CPU capacity of each node is assumed to be equal.

The special case obtained this way is equivalent to the decision version of the bin packing problem. Nodes correspond to bins, with the CPU capacity of the node representing the capacity of the bin. Components correspond to items, with the CPU requirement of the component representing the size of the item. The question whether a given number of bins suffices to hold all items is equivalent to whether the defined problem has a solution.

Since bin packing is strongly NP-hard [25], so is the problem defined here.

3.3 Refined problem model with fog colonies

To allow a decentralized approach to optimization, we assume that the infrastructure consists of multiple parts that can be managed independently. We adopt the notion of fog colonies from [9] to describe such independently manageable parts of the infrastructure.

The set of fog colonies is denoted by K (cf. Table 4). A fog colony $k \in K$ is a set of infrastructure nodes: $k \subseteq V_{\text{HW}}$. The union of all fog colonies is the whole set of infrastructure nodes, i.e., each infrastructure node is contained in at least one fog colony: $n \in V_{\text{HW}} \Rightarrow \exists k \in K, n \in k$. For an infrastructure node $n \in V_{\text{HW}}$, the set of fog colonies containing n is denoted by $K(n) = \{k \in K : n \in k\}$.

We assume that most fog nodes belong to exactly one fog colony, so that both the size and number of fog colonies can be kept as low as possible. If a fog node n belongs to exactly one fog colony, then this fog colony is denoted by k_n.

Baresi et al. suggested that there may be some small overlaps between colonies, i.e., there could be a small number of fog nodes that belong to two fog colonies [17].

In addition to the split of the infrastructure into fog colonies, we assume that also the applications are assigned to fog colonies. Each application is assigned to exactly one fog colony, which should decide on the placement of the given application. The fog colony assigned to the application that component $v \in V_{\text{SW}}$ belongs to is denoted by k_v. Usually, the nodes in the assigned fog colony will host most components of the application, but some components of the application may be transferred to other fog colonies.

4 Solution approaches

Our aim is to compare different decentralization and coordination models. For this, we use in each case the same placement algorithms, described in Section 4.1. Section 4.2 describes the decentralization and coordination models.

4.1 Placement algorithms

We consider two representatives of the typical types of placement algorithms that have been suggested previously in the literature: an exact algorithm and a heuristic [8].

ILP-based algorithm. Our first algorithm uses Integer Linear Programming (ILP). It is based on an ILP formulation of the problem defined in Section 3.1. The algorithm first converts the problem instance to be solved into the corresponding integer linear program, i.e., into a set of linear inequalities over a set of variables. Then, an external ILP solver is used to solve the integer linear program. If the solver finds a solution to the integer linear program, then the new placement of the components is retrieved from this solution and the components are placed or migrated accordingly. It is possible that there exists no solution, e.g., because the infrastructure does not have sufficient capacity to satisfy the needs of all applications. It is also possible that the problem is solvable, but the resources available to the solver do not allow it to find a solution. For instance, the solver can be invoked with a timeout or with a memory limit. In such cases, the placement is unsuccessful.

Search-based algorithm. Our second algorithm uses a problem-specific search strategy to try to construct a solution, i.e., to find a host for all newly added components. The algorithm sorts the newly added components in ascending order of their distance from end devices, and processes the components in this order. The potential hosts (cloud and fog nodes) are also sorted in ascending order of their distance from the end devices. The algorithm tries to place each component on the first host that can accommodate it, taking into account all constraints. If no suitable host could

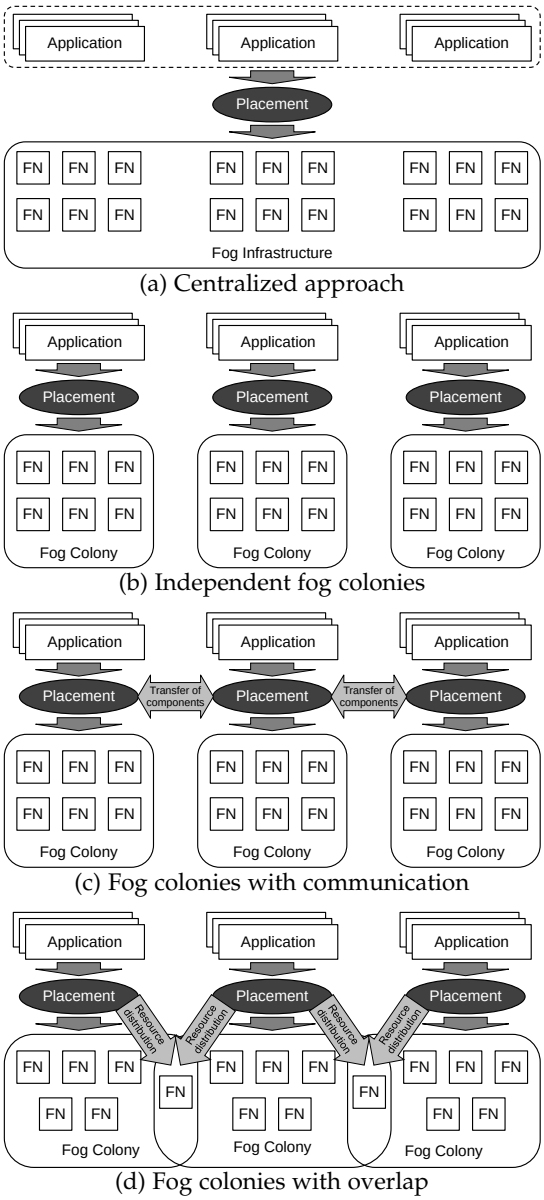


Fig. 1: Schematic overview of the considered decentralization and coordination models. For simplicity, end devices and the cloud are not shown. FN: fog node

be found for a component, then the algorithm checks if the migration of an already placed component to a different host would allow to place the new component.

More details about the two algorithms are available in the supplementary material.

4.2 Decentralization and coordination models

We consider four different approaches for the decentralization and coordination of decision-making, as shown schematically in Fig. 1 and described in the following.

4.2.1 Centralized approach

In the centralized approach (Fig. 1a), decision-making is performed by a central entity that has full knowledge about the entire infrastructure and all applications. The optimization procedure (Algorithm 1) is invoked at regular time intervals or when a change requires a reaction (e.g., a new application has to be placed). To enable optimization, first all necessary information is gathered from the entire system (line 1). This consists of information about all applications that are already placed or that need to be placed, including the requirements of the components and connectors as well as the current placement. Based on this information, the placement algorithm (which can be either the ILP-based or the search-based algorithm) is run (line 2). If the algorithm managed to find a solution, then the new placement of the components is retrieved and the components are placed or migrated accordingly (lines 3-4).

Algorithm 1 Centralized approach

1: Gather information from the entire system
2: Run placement algorithm for the entire system
3: **if** solution found **then**
4: Enact new placement in the entire system
5: **end if**

Algorithm 2 Independent fog colonies

1: Gather information from the fog colony
2: Run placement algorithm for the fog colony
3: **if** solution found **then**
4: Enact new placement in the fog colony
5: **end if**

4.2.2 Independent fog colonies

In the second approach (Fig. 1b), each fog colony is optimized on its own. As shown in Algorithm 2, the steps are basically the same as in the centralized approach. However, in this case, only the infrastructure and applications belonging to the given fog colony are taken into account. That is, only information related to the given fog colony is collected, and the input to the placement algorithm is limited to the infrastructure and applications of the given fog colony. The new placement is enacted only in the given fog colony.

We assume here that the fog colonies are disjoint. For this reason, each fog colony can be optimized independently from the others. The optimization procedure can be invoked for the different fog colonies at different times (either at regular time intervals, or when there is a change in the given fog colony). The enactment of the optimization result in one fog colony does not have any impact on other fog colonies.

4.2.3 Fog colonies with communication

In the third approach (Fig. 1c), we again use fog colonies that are disjoint. The fog colonies are again optimized independently, but now we allow components that are assigned to a fog colony k to be transferred to a neighboring fog colony k' if necessary. This type of coordination can be realized through communication among the controller nodes of the fog colonies, as suggested by [9].

Given two fog colonies $k_1, k_2 \in K$, we say that they are *neighboring fog colonies* if there are nodes $n_1 \in k_1$ and $n_2 \in k_2$ such that n_1 and n_2 are adjacent, i.e., $n_1 n_2 \in E_{\text{HW}}$.

We assume that neighboring fog colonies can communicate with each other and coordinate their placement decisions. If a component assigned to fog colony k cannot be placed in k because of a lack of resources, then the

Algorithm 3 Fog colonies with communication

1: Gather information from the fog colony
2: Gather information from neighboring fog colonies
3: Run placement algorithm on the collected input
4: **if** solution found **then**
5: Enact new placement in fog colony
6: Send components to neighboring fog colonies if needed
7: **end if**

component can be transferred to a neighboring fog colony to be placed there.

Algorithm 3 shows the optimization procedure for a fog colony k. In this case, information is not only collected from fog colony k (line 1), but also from the neighboring fog colonies (line 2). From neighboring colonies, the same types of information are collected as from colony k: the capabilities of the infrastructure and the requirements and placement of the applications already placed. This information is necessary to ensure that a component is transferred to a neighboring fog colony k' only if the component can indeed be placed in k' without violating any constraint.

Based on the collected information, the placement problem is solved with one of the placement algorithms (line 3). The considered infrastructure consists of the union of k and its neighboring fog colonies. The considered components consist of the union of

- the components currently placed on the nodes in k,
- the components currently placed on the nodes in the neighboring colonies of k,
- the components that are newly assigned to k and not placed yet.

During the optimization of fog colony k, components that are already placed in neighboring fog colonies must not be migrated, as they are under the control of other fog colonies. However, they must be taken into account, for two reasons. First, they reduce the amount of resources in the neighboring fog colonies available to components assigned to k. Second, if there is a connector $uv \in E_{\text{SW}}$ for which u is assigned to k and v was transferred to (and already placed in) a neighboring colony k', then the placement of v impacts how the connector can be routed, which in turn impacts the placement of u.

The two placement algorithms introduced in Section 4.1 need some modifications to cope with the specifics of this coordination model. These modifications are described in the supplementary material.

The results of the placement algorithm are on the one hand enacted in the given fog colony (line 5). On the other hand, components that the solution assigns to a foreign node are transferred to the appropriate neighboring fog colony (line 6), where they are placed accordingly.

4.2.4 Fog colonies with overlap

In the fourth approach (Fig. 1d), inspired by [17], we allow some fog nodes to belong to two fog colonies. Such fog nodes are called "fog nodes in overlaps". The capacity of such a fog node can be dynamically distributed between the two fog colonies containing the given fog node. The idea is that the capacity of a fog node in overlap should be used by the fog colony that needs that capacity more.

Algorithm 4 Fog colonies with overlap

1: Gather information from the fog colony
2: Gather information about foreign components in overlaps
3: Run placement algorithm for the fog colony
4: **if** solution found **then**
5: Enact new placement in fog colony
6: **end if**

Let $n \in V_{\text{HW}}$ be a fog node belonging to fog colonies k_1 and k_2. If, for example, k_1 faces a high load and would thus require n, while k_2 is only lightly loaded and can cope with its load also without using n, then k_1 should be able to use n up to its full capacity. If later the load on k_2 increases while k_1 only partially uses the capacity of n, then k_2 should be able to use the remaining free capacity of n. If the load of k_1 decreases, then it should release n as much as possible so that it can be used by k_2 as needed.

To achieve this dynamic assignment of the capacity of fog nodes in overlaps to the involved fog colonies, we perform the following. When optimizing the placement in colony k, all fog nodes belonging to k are taken into account, including the ones that are shared with other fog colonies. A component placed on a fog node in an overlap may only be migrated if the component is assigned to k. That is, if there is a fog node n belonging to both k and a neighboring colony k', and n hosts a component v which is assigned to k', then v must not be migrated as part of the optimization of k. This is important because v may be connected to another component or end device in k' which is not known by k. When determining the placement of components assigned to k, the fog nodes that are not in overlaps should be preferred, to retain the maximum level of flexibility.

Algorithm 4 shows the steps of the procedure. As in the previous approaches, first information about the fog colony is gathered (line 1). In addition, information about foreign components (i.e., components assigned to other fog colonies) that are placed on fog nodes in overlaps is gathered (line 2). The placement algorithm is run on the collected input (line 3). The solution is processed (lines 4-5), just like in the previously introduced approaches. The solution has to be enacted only in the given fog colony k and for the components assigned to k. There is no adverse impact on other fog colonies or the applications assigned to them.

5 Evaluation

This section describes the empirical experience with the approaches from Section 4. We first describe our implementation of the approaches (Section 5.1), followed by the experimental setup (Section 5.2), the results of the experiments (Section 5.3-5.7), and a summary (Section 5.8).

5.1 Implementation

To experiment with different parameter configurations, a simulation-based approach is appropriate. Although there are some existing simulators for fog computing, none of them supports the kinds of decentralized, coordinated and dynamic placement approaches considered in this paper [26]. Hence, we created our own simulation environment, and implemented the approaches described in Section 4 in

TABLE 5: Parameter values (based on [27])

Node	CPU cap. [MIPS]	RAM [MB]
Cloud	120,000	64,000
Proxy server	60,000	8,000
Edge node (small)	6,750	1,000
Edge node (big)	13,500	2,000
Link	**Bandwidth [Mbps]**	**Latency [ms]**
Cloud – Proxy server	10,000	100
Proxy server – Edge node	10,000	2
Edge node – End device	0.65	100
	1	20
	1,000	50
	1,000	12
Between regions	100	50
Component	**CPU size [MIPS]**	**RAM size [MB]**
	2,500-5,000	500-1,000
Connector	**Bandwidth [Mbps]**	**Latency [ms]**
	0.2-0.6	40-200

this simulation environment. The result is a Java program, which is publicly available[1].

For solving the integer programs, the Gurobi Optimizer[2], version 9.1.2 is used. Gurobi is invoked directly from the Java program using Gurobi's Java API. The experiments were carried out on a laptop computer with Intel i7-1165G7 CPU @ 2.80 GHz and 16 GB RAM, running Windows 10 and Java HotSpot 64-bit VM version 16.0.2.

5.2 Experiment setup

With the experiments, we want to answer two questions:

1) How do the results achieved by the investigated decentralization and coordination approaches compare to each other?
2) How does the answer to the previous question depend on the parameters of the problem and on the used algorithm?

To answer these questions, we perform controlled experiments, comparing the four approaches described in Section 4.2 to each other. The primary criterion of the comparison is the success rate, i.e., the ratio of problem instances that could be solved in a given experiment.

We define a *baseline scenario* for the experiments. We use parameters from [27] because that paper adopted values from real-world technologies (e.g., LTE, WiFi and LPWAN for network connections, node specifications from Intel and Cisco etc.). In our baseline scenario, the infrastructure consists of 5 regions. Each region contains the four-level tree topology from [27], consisting of a cloud, a proxy server, 12 edge nodes, and 48 end devices. The parameters of the nodes and the links among them are shown in Table 5. Half of the edge nodes are big, the other half small. There are four categories of links between edge nodes and end devices, and 25% of such links belong to each category. For each region, two neighboring regions are selected. For this purpose, the regions are numbered from 0 to 4, and region i becomes a neighbor of regions $i+1 \pmod 5$ and $i-1 \pmod 5$. If two regions are supposed to be neighbors, then a random fog node is selected from both regions, and the two fog nodes are connected by a link. Altogether, each region contains 62 nodes, leading to 310 nodes in total.

The applications for the baseline scenario are defined as follows. For each region, 5 applications are created. As in [27], each application is either a "master-workers application" or a "sequential unidirectional dataflow application", each with probability 0.5. Each application consists of 12 components. A randomly selected component of each application is connected to a randomly selected end device in the region. Altogether, 25 applications need to be placed, with 300 components in total.

The baseline scenario starts with an empty infrastructure, i.e., no applications are placed yet. In each phase, one application is placed in each region. Within each phase, the application of each region is placed as a separate optimization step. The scenario ends after 5 phases, when each application has been placed. Each phase consists of 5 steps.

In the centralized approach, the whole infrastructure is considered in each optimization step. In the "independent fog colonies" approach, as well as in the "fog colonies with communication" approach ("communicating" for short), each region is considered a fog colony. In the "fog colonies with overlap" approach ("overlapping" for short), we enlarge the colonies to make neighboring fog colonies overlap. In particular, if fog colonies k_1 and k_2 are neighbors, then we include two random nodes from k_1 in k_2 and two random nodes from k_2 in k_1. While doing so, we ensure that a fog node that is already shared by two fog colonies is not shared with a third one.

The external ILP solver is always invoked with a timeout of 60 seconds. To limit the impact of random effects, we repeat each experiment 30 times.

5.3 Results of the baseline scenario

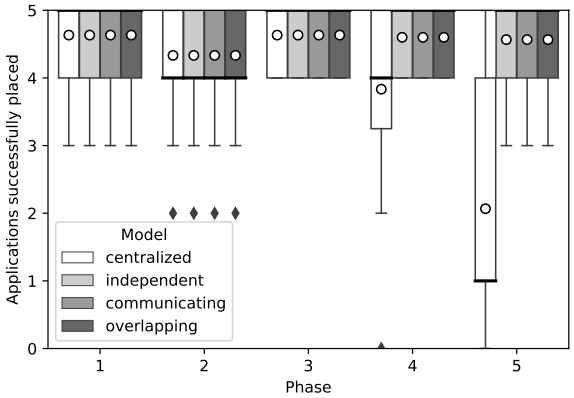
Fig. 2 shows the results of the baseline scenario in terms of the number of successfully placed applications. The values are shown for each phase of the baseline scenario. In this and the next figures, box plots are used to convey the distribution of results. In the box plots, the box goes from the first to the third quartile of the distribution, whereas the whiskers reach to cover the rest of the distribution, except for outliers which are represented as diamonds. The median is shown by a horizontal line and the mean by a circle.

As shown in Fig. 2a, the ILP-based algorithm manages to place most applications in most cases. The four considered models of decentralization and coordination lead to identical results in the first three phases, but the "centralized" approach leads to worse results in the last two phases than the decentralized approaches.

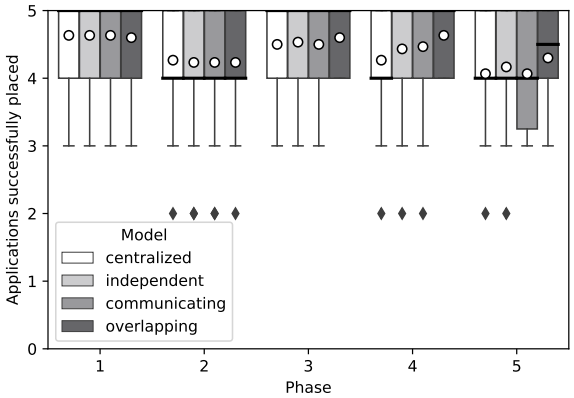
As shown in Fig. 2b, the search-based algorithm returns results comparable to those of the ILP-based algorithm. The differences among the four considered models of decentralization and coordination are higher in this case. In the last three phases, the "overlapping" approach leads to the best results.

In terms of execution time, there are huge differences between the considered approaches. (Details are shown in

1. https://github.com/zoltanmann/fapp-colonies
2. https://www.gurobi.com/

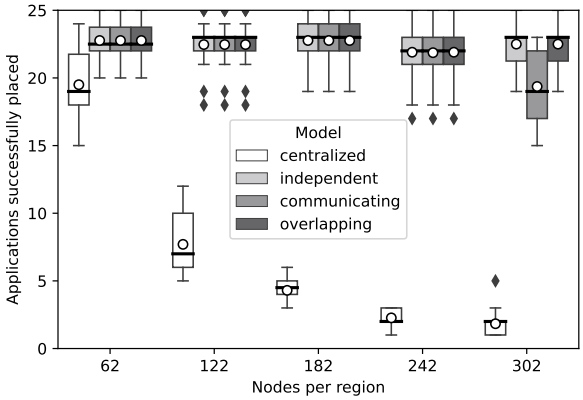


(a) Number of successfully placed applications with the ILP solver

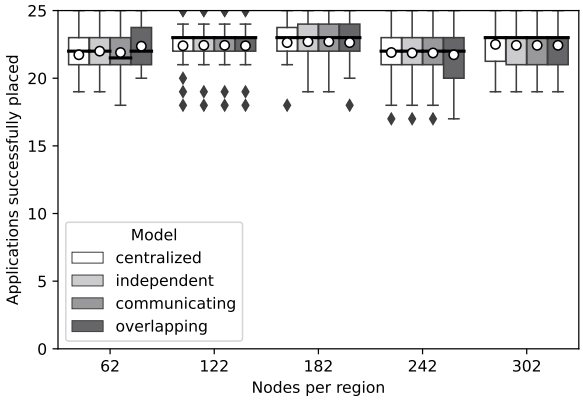


(b) Number of successfully placed applications with the search-based algorithm

Fig. 2: Results of the baseline scenario



(a) Number of successfully placed applications with the ILP solver



(b) Number of successfully placed applications with the search-based algorithm

Fig. 3: Impact of increasing fog colony size

the supplementary material.) The execution time of the search-based algorithm is in all cases below 60 milliseconds. The execution time of the ILP-based algorithm goes up to several minutes for the centralized approach and several seconds for the decentralized approaches.

5.4 Impact of fog colony size

In this and the next two experiments, we vary different parameters of the input one by one. Then, in Section 5.7, we combine all these aspects in a single experiment. Specifically in this experiment, we vary the number of edge nodes per region from 12 until 60 in steps of 12. As a result, the number of all nodes per region varies from 62 to 302, and the number of all nodes in the whole infrastructure from 310 to 1510. All other parameters of the baseline scenario remain the same.

The results are summarized in Fig. 3. A column in Fig. 3 shows the value aggregated over the whole sequence of adding all applications in all regions.

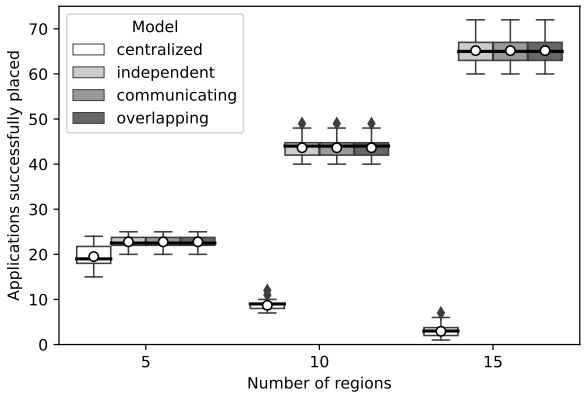
Fig. 3a shows how the number of applications successfully placed by the ILP-based algorithm changes with increasing fog colony size. As the number of nodes per region increases, the centralized approach can place less and less applications. This is because of the limited scalability of the centralized approach. On the other hand, all decentralized approaches can consistently place a high number of applications.

Fig. 3b shows the results of the search-based algorithm. In this case, all centralized and decentralized approaches manage to place a high number of applications.

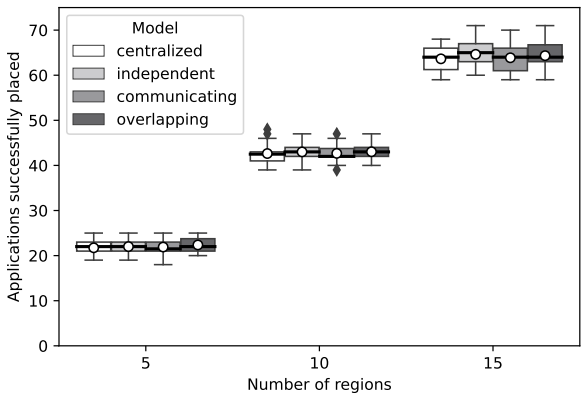
5.5 Impact of the number of fog colonies

In this experiment, we vary the number of regions from 5 until 15 in steps of 5. Since there are 5 applications per region, this leads not only to an increasing infrastructure size, but also to an increasing number of applications. All other parameters of the basic scenario remain the same. For 15 regions, the infrastructure consists of 930 nodes, and there are 75 applications, with 900 components altogether.

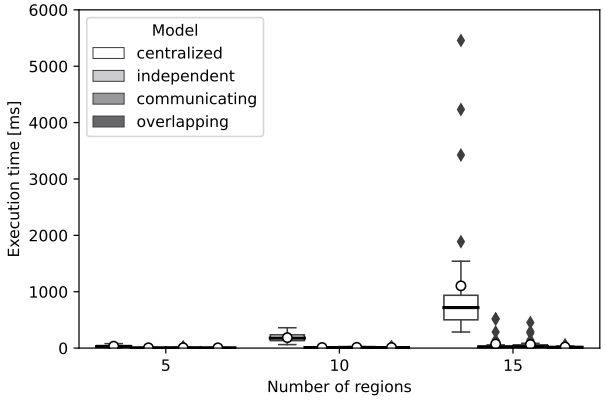
The results are shown in Fig. 4. Concerning the results of the ILP-based algorithm (Fig. 4a), the scalability issue of the centralized approach becomes again apparent, just as in the previous experiment. For 10 and 15 regions, only a fraction of the applications could be placed with the centralized approach. On the other hand, the decentralized approaches manage to place a similarly high number of applications.



(a) Number of successfully placed applications with the ILP solver

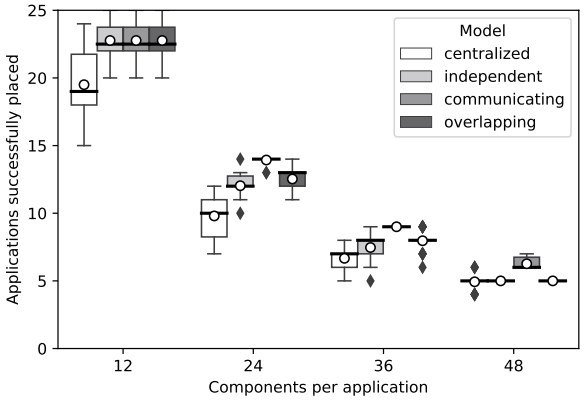


(b) Number of successfully placed applications with the search-based algorithm

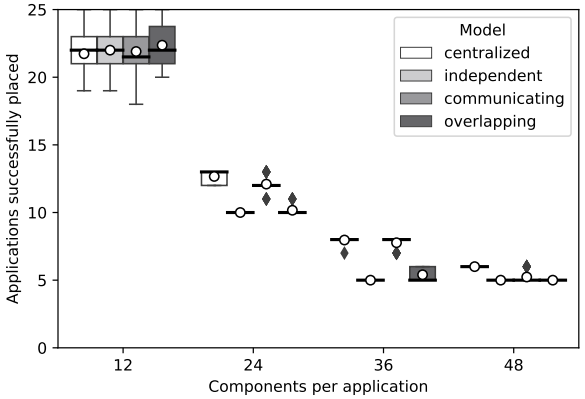


(c) Execution time of the search-based algorithm

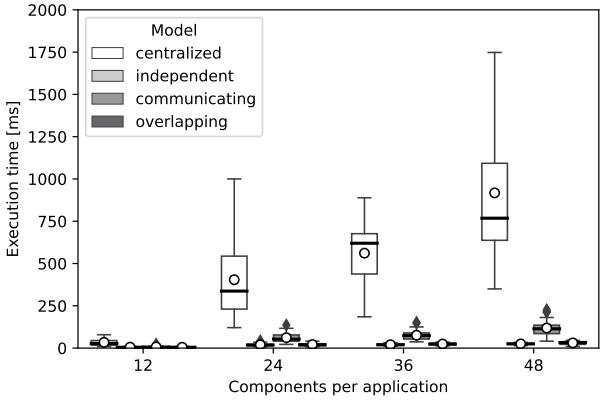
Fig. 4: Impact of increasing the number of fog colonies



(a) Number of successfully placed applications with the ILP solver



(b) Number of successfully placed applications with the search-based algorithm



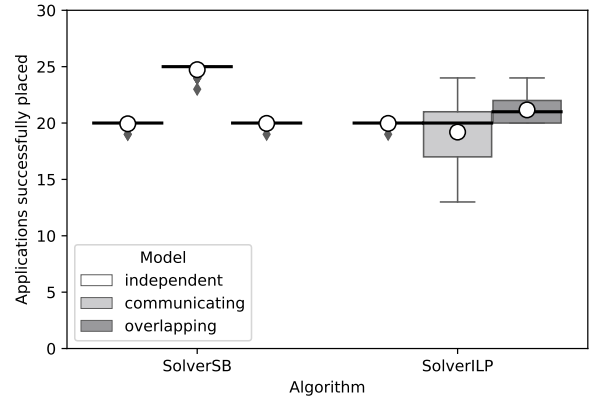
(c) Execution time of the search-based algorithm

Fig. 5: Impact of increasing the number of components per application

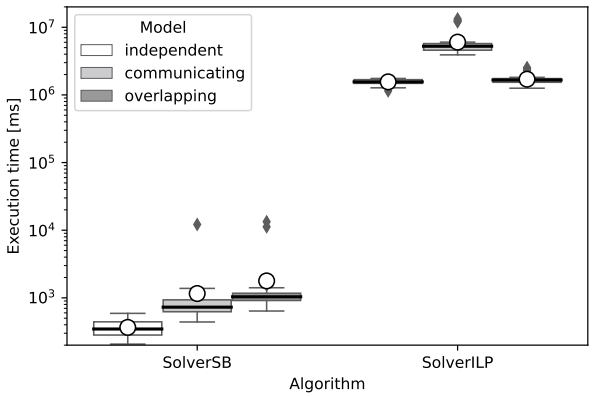
With the search-based algorithm, all four approaches lead to a similarly high number of placed applications (Fig. 4b). However, looking also at the execution time of the search-based algorithm (Fig. 4c), the problems of the centralized approach start to arise here as well. The rapid growth of the execution time suggests that scalability could become a problem for even larger systems.

5.6 Impact of the load

In this experiment, we vary the number of components per application from 12 until 48 in steps of 12. This way, the load on the overall fog system increases, making it increasingly harder to place the applications. All other parameters of the basic scenario remain the same.



(a) Number of successfully placed applications



(b) Execution time (logarithmic scale)

Fig. 6: Results of the large-scale experiment

The results are shown in Fig. 5. As Fig. 5a shows, for higher loads (24 or more components per application), the ILP-based algorithm achieves its best results consistently with the "communicating" approach. For the search-based algorithm, the "centralized" and the "communicating" approaches lead to the best results for 24 or more components per application (Fig. 5b). On the other hand, Fig. 5c reveals that the "communicating" approach requires significantly less time than the "centralized" approach.

5.7 Large-scale experiment

Finally, we combined the different dimensions of scaling of the previous experiments in a single experiment. In this case, we created 10 regions, each containing 242 nodes, leading to 2,420 infrastructure nodes altogether. Each application consists of 48 components, leading to 2,400 components altogether (10 regions, 5 applications per region, 48 components per application). The number of infrastructure nodes and components considered in this experiment is higher than what was considered in most of the previous papers on fog application placement algorithms [8].

The results are shown in Fig. 6. The centralized approach was skipped because we knew from the earlier experiments that it is not appropriate for such large and hard problem instances. Fig. 6a shows that coordination has the potential to improve the results. For the search-based algorithm (denoted as "SolverSB" in the figure), the "communicating" approach leads to 25% better results than the "independent" approach, while the "overlapping" approach leads to no improvement. For the ILP-based algorithm (denoted as "SolverILP" in the figure), the "overlapping" approach leads to about 5% better results on average than the "independent" approach. On the other hand, coordination also leads to non-negligible overhead in terms of execution time, as shown in Fig. 6b.

5.8 Summary

The insights from the experimental results can be summarized as follows. For small infrastructures, all considered decentralization and coordination models lead to similar results. This is because these are easy problem instances, for which optimal or near-optimal solutions can be found in all considered decentralization and coordination models.

As either the number of regions or the number of nodes per region increases, scalability of the ILP-based algorithm in the centralized model becomes problematic. In these cases, the solution space in the centralized model becomes huge, so that the ILP solver can only search a tiny fraction of it in the given time period [28]. The ILP-based algorithm with any of the decentralized models seems to scale well. This means that good solutions can be found locally, in the given fog colony, without a need for global load balancing. Also, the decentralized models lead to a smaller solution space, which is manageable for the ILP solver.

The search-based algorithm scales well with all four considered models. This is probably due to the problem-specific sorting strategies for components and nodes used in the search-based algorithm, leading to good solutions through limited search effort. However, the execution time of the search-based algorithm quickly increases in the centralized model. This is because the number of potential hosts considered for each component becomes large. Nevertheless, the execution time of the search-based algorithm stays within a couple of seconds in even the largest experiments.

The effect of coordination between colonies becomes only important for the placement of large applications. With large applications, it happens more frequently that a fog colony gets overloaded and can thus benefit from coordination with neighboring fog colonies. For large applications, especially the "communicating" coordination model proved useful, although it leads to higher execution time than the "independent" model. The "communicating" coordination model leads to the highest number of placement options outside the given region, which explains both the good results achieved and the relatively high execution time.

6 Discussion of possible extensions

This section discusses some potential extensions and generalizations of the work presented in this paper.

6.1 Handling other types of changes

To foster easy interpretability, our experiments were focused on one type of change: adding new applications. There can

also be other types of relevant changes: e.g., applications can be removed, the resource requirements of components can change, new infrastructure nodes can become available etc. The approaches and experiments presented in this paper can be easily extended to account for such types of changes. This is because the considered algorithms always work with the current state of the infrastructure and of the applications. Thus, upon any kind of change, the algorithms can be re-run to react to the change, as in [29].

6.2 Other types of decentralization and coordination

In this paper, we focused on four models of decentralization and coordination. These are the ones that, to our knowledge, have been proposed in the context of fog computing. In other distributed computing paradigms, which have a longer history than fog computing, such as grid, cloud and cluster computing, also other types of decentralization and coordination approaches have been proposed. Examples include dynamic partitioning of resources between multiple schedulers [30], shared-state scheduling, in which multiple schedulers have simultaneous access to the same set of resources [31], and hierarchical resource management approaches [32], [33]. In future work, it would be interesting to investigate whether these approaches could be successfully adopted in the context of fog computing, and if yes, how they compare to the approaches evaluated in this paper.

6.3 Other effects of decentralization and coordination

When comparing different decentralization and coordination approaches, we focused on algorithm execution time and the quality of the results. There are also some further aspects that could be investigated. For example, decentralized approaches may offer better *fault tolerance*, especially if neighboring fog colonies coordinate to help out in the event of a failure in one of the fog colonies. However, coordination among fog colonies increases *complexity*. This holds both for the "communicating" and the "overlapping" coordination approaches. Complexity can make a practical implementation more difficult and also more error-prone. In addition, coordination may also lead to an increased *lead time* in the deployment of new components. This holds in particular for the "communicating" coordination approach: if a component is dispatched to a neighboring fog colony, this increases the time needed to deploy the given component [13].

6.4 Metrics in the optimization problem

Different variants of the fog application placement problem use different metrics to quantify the suitability of potential solutions. Metrics could include for example latency, resource demand, energy consumption, financial costs etc. Each metric may be subject to a constraint or used as an optimization objective [34].

In this paper, we used resource consumption (CPU, RAM, bandwidth), latency, and the number of migrations as metrics. Future work could extend our investigations to other metrics, such as energy consumption. Such extension is straight-forward as long as the number of optimization objectives is at most one, because both of the considered algorithms can easily accommodate additional constraints (for the ILP-based algorithm, this only holds true if the metric can be calculated by means of linear equations and inequalities). If multiple metrics should be optimized, then the handling of their relative importance adds a new level of complexity [35]. For this purpose, different techniques from the rich theory of multi-objective optimization could be applied [36].

6.5 Application priorities

In the problem considered in this paper, all applications are assumed to be equally important. A possible extension could be to consider applications with different priorities [37]. If a new application with high priority cannot be placed, an already placed application with lower priority could be paused to temporarily free some capacity for the new application. The paused application could be resumed once the application with higher priority has finished its execution. Keeping track of paused applications would add some additional complexity, which, however, is mostly orthogonal to the main objective of this work regarding decentralization and coordination.

7 Conclusions and Future Work

This paper investigated the effects of decentralization and coordination in fog application placement. For this purpose, we implemented two fog application placement algorithms in conjunction with four different decentralization and coordination models. The approaches were compared by means of extensive experiments. We found that the centralized approach achieves good results on relatively small and easy problem instances. However, as the size of the infrastructure and/or the number of components to be placed increases, the results of the centralized approach deteriorate heavily, while all the considered decentralized approaches scale much better. Among the decentralized approaches, we could observe that coordination among the fog colonies leads to improved results for the hardest problem instances. The "fog colonies with communication" approach proved particularly successful in terms of the number of placed applications, although it increases execution time.

While this work has shown the fundamental advantage of coordinated decentralized approaches for fog application placement as well as the impact of different coordination mechanisms, the question of which is the best coordination approach remains open. Although the "fog colonies with communication" approach seems best based on our experiments, further research should investigate to what extent this also holds in other settings (e.g., with other types of constraints and optimization objectives, taking also into account the considerations of Section 6). It would be important to validate the findings also in real fog environments. Also other coordination mechanisms should be investigated, for example based on the coordination patterns identified in previous work [38].

Acknowledgments

This work was partially supported by the EU's Horizon 2020 research and innovation programme under grant agreement

no. 871525 (FogProtect). The author wishes to thank Patrick Kuhs for help with a preliminary implementation.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, 2012, pp. 13–16.

[2] 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.

[3] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[4] Z. Á. Mann, "Notions of architecture in fog computing," *Computing*, vol. 103, no. 1, pp. 51–73, 2021.

[5] 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.

[6] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Transactions on Internet Technology*, vol. 19, no. 1, pp. 1–21, 2018.

[7] 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.

[8] S. Smolka and Z. Á. Mann, "Evaluation of fog application placement algorithms: A survey," *Computing*, accepted, 2021.

[9] 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.

[10] 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.

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

[12] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for IoT services in the fog," in *IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016, pp. 32–39.

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

[14] Q. T. Minh, D. T. Nguyen, A. Van Le, H. D. Nguyen, and A. Truong, "Toward service placement on fog computing landscape," in *4th NAFOSTED Conference on Information and Computer Science*. IEEE, 2017, pp. 291–296.

[15] A. A. Alsaffar, H. P. Pham, C.-S. Hong, E.-N. Huh, and M. Aazam, "An architecture of IoT service delegation and resource allocation based on collaboration between fog and cloud computing," *Mobile Information Systems*, vol. 2016, 2016.

[16] M. M. Shurman and M. K. Aljarah, "Collaborative execution of distributed mobile and IoT applications running at the edge," in *International Conference on Electrical and Computing Technologies and Applications (ICECTA)*. IEEE, 2017.

[17] L. Baresi, D. F. Mendonça, and G. Quattrocchi, "PAPS: A framework for decentralized self-management at the edge," in *17th International Conference on Service-Oriented Computing (ICSOC)*. Springer, 2019, pp. 508–522.

[18] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *35th Annual IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2016.

[19] G. Lee, W. Saad, and M. Bennis, "An online secretary framework for fog network formation with minimal latency," in *IEEE International Conference on Communications (ICC)*. IEEE, 2017.

[20] ——, "An online optimization framework for distributed fog network formation with minimal latency," *IEEE Transactions on Wireless Communications*, vol. 18, no. 4, pp. 2244–2258, 2019.

[21] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: Towards minimizing delay in the Internet of Things," in *IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2017, pp. 17–24.

[22] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, 2018.

[23] C. Guerrero, I. Lera, and C. Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 6, pp. 2435–2452, 2019.

[24] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 516–529, 2018.

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

[26] C. Kunde and Z. Á. Mann, "Comparison of simulators for fog computing," in *35th Annual ACM Symposium on Applied Computing (SAC)*, 2020, pp. 1792–1795.

[27] T. Djemai, P. Stolf, T. Monteil, and J.-M. Pierson, "A discrete particle swarm optimization approach for energy-efficient IoT services placement over fog infrastructures," in *18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2019, pp. 32–40.

[28] Z. Á. Mann, *Optimization in computer engineering–Theory and applications*. Scientific Research Publishing, Inc. USA, 2011.

[29] Z. Á. Mann, A. Metzger, J. Prade, and R. Seidl, "Optimized application deployment in the fog," in *17th International Conference on Service-Oriented Computing (ICSOC)*. Springer, 2019, pp. 283–298.

[30] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *8th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2011, pp. 295–308.

[31] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *8th ACM European Conference on Computer Systems*, 2013, pp. 351–364.

[32] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, and L. Zhang, "A hierarchical approach for the resource management of very large cloud platforms," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 253–272, 2013.

[33] M. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth, "mck8s: An orchestration platform for geo-distributed multi-cluster environments," in *30th International Conference on Computer Communications and Networks (ICCCN)*, 2021.

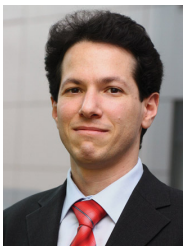
[34] Z. Á. Mann, "Optimization problems in fog and edge computing," in *Fog and Edge Computing: Principles and Paradigms*. Wiley, 2019, pp. 103–121.

[35] T. Menouer, "KCSS: Kubernetes container scheduling strategy," *The Journal of Supercomputing*, vol. 77, no. 5, pp. 4267–4293, 2021.

[36] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[37] C. Zhang, H. Tan, H. Huang, Z. Han, S. H.-C. Jiang, N. Freris, and X.-Y. Li, "Online dispatching and scheduling of jobs with heterogeneous utilities in edge computing," in *21st International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2020, pp. 101–110.

[38] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka, "On patterns for decentralized control in self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 76–107.



Zoltán Ádám Mann is associate professor at the University of Amsterdam, The Netherlands. He holds a PhD in Computer Science from Budapest University of Technology and Economics, Hungary. His research interests include fog computing, cloud computing, and optimization algorithms.

Decentralized application placement in fog computing – Supplementary material –

Zoltán Ádám Mann

In this supplementary file, we give a detailed description of the two placement algorithms as well as their usage in conjunction with the four considered models of decentralization and coordination. In addition, diagrams about the execution time of the algorithms in the experimental evaluation are shown.

1 Placement algorithms

1.1 Integer Linear Programming formulation

We first devise an Integer Linear Programming (ILP) formulation of the problem defined in Section 3 of the paper.

To encode the functions α and β, we define two sets of binary variables (see also Table 1). Specifically, α is encoded using the variables $\{x_{v,n} : v \in V_{\text{SW}}, n \in V_{\text{HW}}\}$, where $x_{v,n} = 1$ if and only if component v is to be placed on node n. To allow a uniform handling of components and end devices, we extend $x_{v,n}$ to $v \in \overline{V_{\text{SW}}}, n \in \overline{V_{\text{HW}}}$: for $v \in \mathcal{E}$ and $n \in \overline{V_{\text{HW}}}$, we let $x_{v,n} = 1$ if and only if $n = v$. The function β is encoded using the variables $\{y_{e,p} : e \in E_{\text{SW}}, p \in P\}$, where $y_{e,p} = 1$ if and only if connector e is to be routed through path p.

For the placement to be valid, we have to ensure that each component is placed on exactly one infrastructure node and not on an end device, and each connector is routed through exactly one path. For this reason, we introduce the following sets of equations:

$$\forall v \in V_{\text{SW}} : \sum_{n \in V_{\text{HW}}} x_{v,n} = 1, \tag{9}$$

$$\forall v \in V_{\text{SW}}, \forall n \in \mathcal{E} : \; x_{v,n} = 0, \tag{10}$$

$$\forall e \in E_{\text{SW}} : \sum_{p \in P} y_{e,p} = 1. \tag{11}$$

TABLE 1: Variables (all of them are binary)

Name	Description
$x_{v,n}$	1 iff component $v \in V_{\text{SW}}$ is to be placed on node $n \in V_{\text{HW}}$
$y_{e,p}$	1 iff connector $e \in E_{\text{SW}}$ is to be routed through path $p \in P$
z_v	1 iff component $v \in V_{\text{SW}}^0$ has to be migrated

In addition, we have to ensure that each end device is placed on itself:

$$\forall v \in \mathcal{E} : \; x_{v,v} = 1, \tag{12}$$

$$\forall v \in \mathcal{E}, \forall n \in \overline{V_{\text{HW}}} \setminus \{v\} : \; x_{v,n} = 0. \tag{13}$$

The consistency of placement and routing (equation (3) in the paper) must be ensured in terms of the newly introduced variables. The $x_{v,n}$ and $y_{e,p}$ variables must be assigned consistent values, so that, if $vv' \in E_{\text{SW}}$, v is placed on n and v' is placed on n', then the connector vv' is routed through a path between n and n':

$$\forall vv' \in E_{\text{SW}}, \forall n \in \overline{V_{\text{HW}}}, \forall n' \in \overline{V_{\text{HW}}} : \sum_{p \in P_{n,n'}} y_{vv',p} \geq x_{v,n} + x_{v',n'} - 1. \tag{14}$$

Equation (14) may require some explanation. It should be noted that if at least one of $x_{v,n}$ and $x_{v',n'}$ is 0, then the inequality does not represent a constraint since the left-hand side is non-negative and the right-hand side is non-positive. On the other hand, if $x_{v,n} = x_{v',n'} = 1$, that is, if v is placed on n and v' is placed on n', then (14) constrains the sum on the left-hand side to be at least 1. Because of (11), the sum on the left-hand side of (14) cannot be more than 1, so it has to be exactly 1, which is what we wanted to express.

The capacity constraints of the infrastructure nodes (equations (4)-(5) in the paper) are represented in terms of the variables by the following inequalities:

$$\forall n \in V_{\text{HW}} : \sum_{v \in V_{\text{SW}}} x_{v,n} \cdot c_v \leq C_n, \tag{15}$$

$$\forall n \in V_{\text{HW}} : \sum_{v \in V_{\text{SW}}} x_{v,n} \cdot r_v \leq R_n. \tag{16}$$

The bandwidth constraints of the infrastructure links (equation (6) in the paper) are ensured in terms of the variables by the following inequality:

$$\forall l \in E_{\text{HW}} : \sum_{e \in E_{\text{SW}}} \sum_{p \in P(l)} y_{e,p} \cdot b_e \leq B_l. \tag{17}$$

The latency constraints of the connectors (equation (7) in the paper) are ensured in terms of the variables by the following inequality:

$$\forall e \in E_{\text{SW}} : \sum_{p \in P} y_{e,p} \cdot D(p) \leq d_e. \tag{18}$$

Explanation of (18): because of (11), exactly one of the terms in the sum in (18) will be non-zero, namely the term corresponding to the path through which e is routed. That is, the value of the sum in (18) will be exactly the latency of the path through which e is routed.

To capture migrations, we introduce further binary variables $\{z_v : v \in V^0_{\text{SW}}\}$, where $z_v = 1$ if and only if component v is to be migrated. The z_v variables are determined by the $x_{v,n}$ variables, as captured by the following equation:

$$\forall v \in V^0_{\text{SW}} : \; z_v = 1 - x_{v,h_v}. \tag{19}$$

The objective function can now be easily formulated:

$$\text{minimize} \sum_{v \in V^0_{\text{SW}}} z_v. \tag{20}$$

To sum up, we have to find values for the variables that satisfy equations (9)-(19) while optimizing (20).

1.2 Search-based algorithm

The pseudo-code of this algorithm is given in Algorithm 1. The algorithm aims to place the newly added components, which are stored in a list $\mathcal{N}$ (line 1). The set of end devices that are connected to at least one of the newly added components is determined and denoted $\mathcal{D}$ (line 2). The cloud and fog nodes that can potentially host the newly added components are stored in a list $\mathcal{H}$ (line 3).

Breadth-first search is used in the application graph to determine the distance of the components in $\mathcal{N}$ from the end devices in $\mathcal{D}$. Similarly, breadth-first search is used in the infrastructure graph to determine the distance of the nodes in $\mathcal{H}$ from the end devices in $\mathcal{D}$. On the basis of this information, both $\mathcal{N}$ and $\mathcal{H}$ are sorted in ascending order of distance from $\mathcal{D}$ (lines 4-5).

After these pre-processing steps, the algorithm iterates over the components in $\mathcal{N}$ in the given order (lines 6-41). For each component v, the algorithm tries to find a host by checking the nodes in $\mathcal{H}$ in the given order (lines 8-14). If a suitable host is found, then v is placed on it and the algorithm continues with the next component (lines 9-13).

If no suitable host could be found for v, then the algorithm checks if a host could be relieved by a migration (lines 15-36). For this purpose, the algorithm iterates through all movable components (lines 16-35). For each movable component v', the algorithm iterates over all hosts in $\mathcal{H}$ to check if one of them would be a suitable migration target (lines 19-24). If a migration target for v' could be identified, then v' is migrated and it is checked whether the new component v can be placed on the old host of v' (lines 25-34). If the migration does not allow the placement of v, then it is undone (line 32).

If none of the considered migrations allowed the placement of v, then the placement of any already placed components of the same application is undone (lines 37-40).

Algorithm 1 Search-based placement algorithm

```
 1: N ← {newly added components}
 2: D ← {end devices connected to components in N}
 3: H ← {available fog and cloud nodes}
 4: Sort N in ascending order of distance from D
 5: Sort H in ascending order of distance from D
 6: for v ∈ N do
 7:     success ← false
 8:     for h ∈ H do
 9:         if v can be placed on h then
10:             place v on h
11:             success ← true
12:             break
13:         end if
14:     end for
15:     if success == false then
16:         for v' ∈ {movable components} do
17:             h0 ← current host of v'
18:             h1 ← null
19:             for h' ∈ H do
20:                 if h' ≠ h0 and v' can be migrated to h' then
21:                     h1 ← h'
22:                     break
23:                 end if
24:             end for
25:             if h1 ≠ null then
26:                 migrate v' to h1
27:                 if v can be placed on h0 then
28:                     place v on h0
29:                     success ← true
30:                     break
31:                 else
32:                     undo migration
33:                 end if
34:             end if
35:         end for
36:     end if
37:     if success == false then
38:         undo placement of application
39:         break
40:     end if
41: end for
```

1.3 Complexity

Next, we analyze the computational complexity of the algorithms outlined above.

The ILP-based algorithm consists of creating the integer program, running the external ILP solver, and retrieving the solution from the solver's output. It can be seen easily that creating the integer program and retrieving the solution can be done in polynomial time. However, the ILP solver has an exponential worst-case complexity.

To analyze the complexity of the search-based algorithm, we first look at the non-trivial atomic operations in Algorithm 1. These are operations of the types "if v can be placed on h", "place v on h", "v can be migrated to h", "migrate v to h" and "undo migration", where v is a component and h is a host. These are non-trivial because they not only involve the placement of a component (which would take $O(1)$ steps), but also the routing of all connectors incident to the component. The latter also involves an update of free bandwidth along the path through which the connector is routed. Thus, these operations take $O(\Delta \cdot \Pi)$ time, where Δ is the maximum number of connectors incident to a component and Π is the maximum path length.

The complexity of Algorithm 1 can now be determined by looking at its nested loop structure. The loops of lines 8-14 and 19-24 take $O(|\mathcal{H}| \cdot \Delta \cdot \Pi)$ time. Thus, the loop of lines 16-35 takes $O(|E_{\text{SW}}| \cdot |\mathcal{H}| \cdot \Delta \cdot \Pi)$ time. Therefore, the computational complexity of the search-based algorithm is $O(|\mathcal{N}| \cdot |E_{\text{SW}}| \cdot |\mathcal{H}| \cdot \Delta \cdot \Pi)$.

2 Decentralization and coordination models

Here, we describe how the two placement algorithms are modified to take into account the considered models of decentralization and coordination. This is based on a categorization of nodes and of components.

2.1 Categorization of nodes

The nodes are categorized as follows:

- *Freely usable* nodes can be preferentially used to host components.
- *Unpreferred* nodes should only be used to host components if necessary.

Depending on the considered model of decentralization and coordination, the categories of nodes are determined as follows:

- Centralized: every node is freely usable, no node is unpreferred.
- Independent: every node in the fog colony is freely usable, no node is unpreferred.
- Communicating: every node in the fog colony is freely usable, every node in the neighboring colonies is unpreferred.
- Overlapping: every node in the fog colony that is not shared with another colony is freely usable, every node in the fog colony that is shared with another colony is unpreferred.

Nodes not mentioned above are simply not considered. For example, in the "independent" model, nodes outside the given colony are not considered at all.

2.2 Categorization of components

The components are categorized as follows:

- *New* components are the components of newly added applications that are not placed yet.
- *Fully controlled* components are already placed components whose placement is under the control of the given algorithm invocation.
- *Obtained* components are already placed components that the given colony got from another colony.
- *Read-only* components are already placed components in a neighboring colony that have a connector to a component in the given colony.

Depending on the considered model of decentralization and coordination, the categories of the already placed components are determined as follows:

- Centralized: Each already placed component is fully controlled. There are no obtained or read-only components.
- Independent: Each already placed component assigned to the given colony is fully controlled. There are no obtained or read-only components.
- Communicating: Each component placed in the given colony and assigned to the given colony is fully controlled. Each component placed in the given colony and assigned to a different colony is obtained. Each component placed in a neighboring colony and connected to a component placed in the given colony is read-only.
- Overlapping: Each component placed in the given colony is fully controlled. There are no obtained or read-only components.

Also here, components not mentioned above are disregarded. For example, in the "independent" model, components placed in other colonies are not considered.

2.3 Modifications of the algorithms

To handle the different decentralization and coordination models, some specific constraints are necessary. For the ILP-based algorithm, these are added as additional linear equations or inequalities. For the search-based algorithm, the constraints are added as additional checks in the routines that determine if a given component can be placed on or migrated to a given node (called in lines 9, 20 and 27 of Algorithm 1). Specifically, the following constrains are added:

- Read-only components must not be migrated.
- A component that colony k obtained from colony k' may only be placed in k or k'. (This is important because otherwise a situation could arise in which components of an application would be in non-neighboring colonies k' and k''. In this case, it would be unknown in colony k' what can be done with the component in k'.)
- If there is a connector v_1v_2, and k' and k'' are colonies different from each other and from ours, then it is forbidden to place v_1 on k' and v_2 on k''. (The reason is similar as before.)

The handling of unpreferred nodes is different in the two algorithms. In the ILP-based algorithm, a penalty term is added to the objective function that penalizes the usage of unpreferred nodes. In the search-based algorithm, the sorting of nodes (line 5 in Algorithm 1) is modified, such that freely usable nodes are first, and the unpreferred nodes are at the end of list.

3 Additional experimental results

Because of the space limitation, Section 5 of the paper focused on the number of applications successfully placed in the different models of decentralization and coordination, and the execution time was only shown in some cases. For the sake of completeness, here we include execution time diagrams of each experiment of Sections 5.3-5.6 of the paper. It should be noted that, to improve readability, the execution time diagrams of the ILP-based algorithm use logarithmic scale and do not show averages.

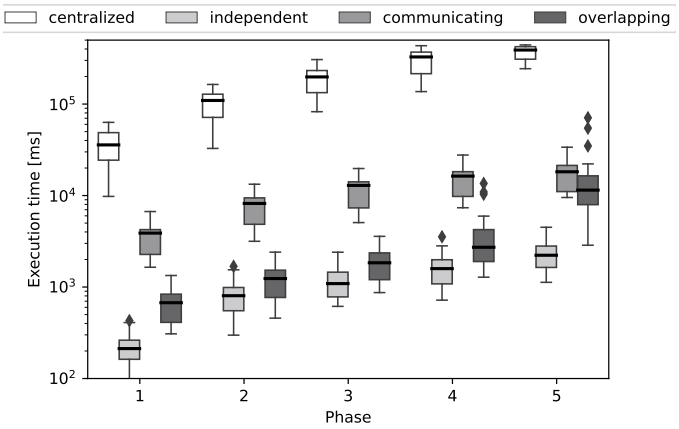


Fig. 1: Execution time of the ILP-based algorithm in the baseline scenario (logarithmic scale)

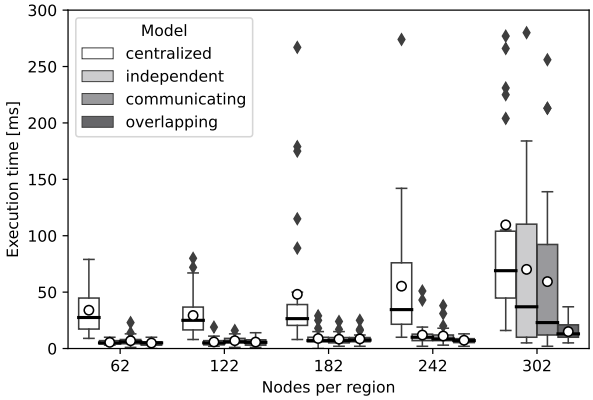


Fig. 4: Execution time of the search-based algorithm for varying fog colony size

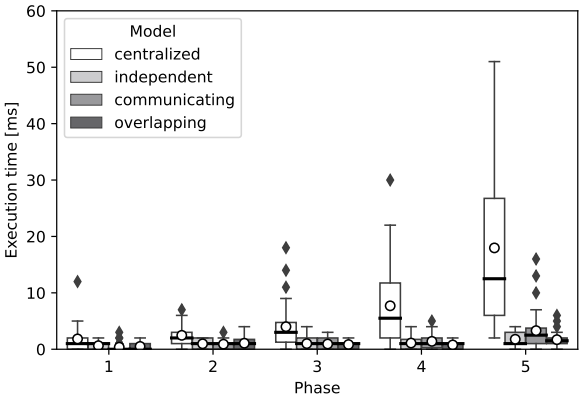


Fig. 2: Execution time of the search-based algorithm in the baseline scenario

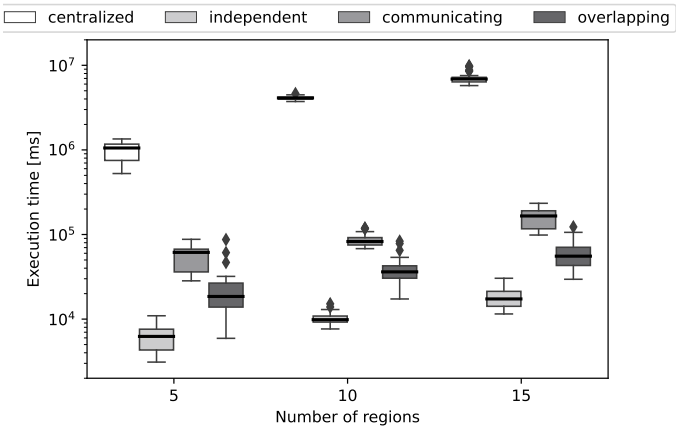


Fig. 5: Execution time of the ILP-based algorithm for varying number of fog colonies (logarithmic scale)

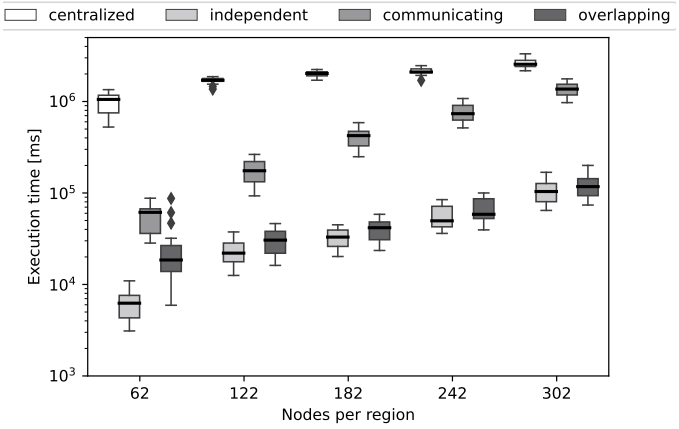


Fig. 3: Execution time of the ILP-based algorithm for varying fog colony size (logarithmic scale)

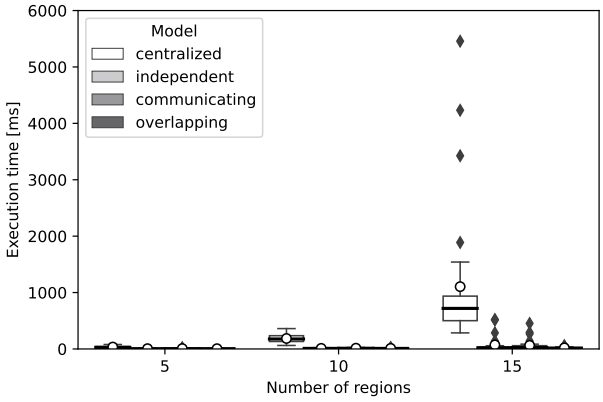


Fig. 6: Execution time of the search-based algorithm for varying number of fog colonies

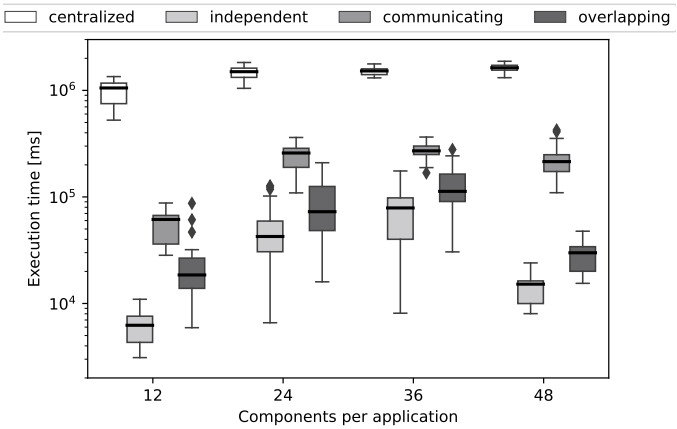


Fig. 7: Execution time of the ILP-based algorithm for varying application size (logarithmic scale)

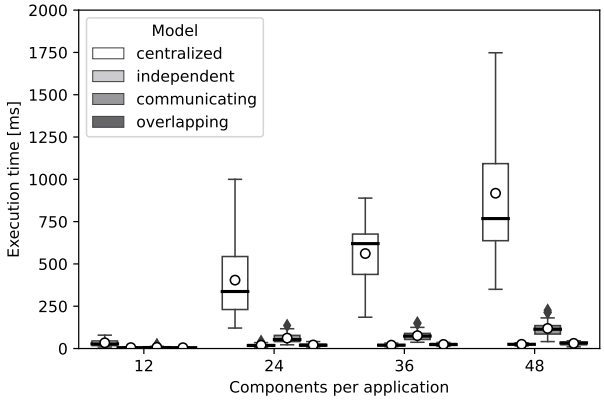


Fig. 8: Execution time of the search-based algorithm for varying application size