Fog Application Allocation for Automation Systems

Marco Suter
ETH Zurich
Switzerland
marco.suter5@alumni.ethz.ch

Raphael Eidenbenz
ABB Corporate Research
Switzerland
raphael.eidenbenz@ch.abb.com

Yvonne-Anne Pignolet†
DFINITY
Switzerland
yvonneanne@dfinity.org

Ankit Singla
ETH Zurich
Switzerland
ankit.singla@inf.ethz.ch

***Abstract*—Fog computing brings the convenience of cloud computing closer to the edge of the Internet and even further into local area networks. A central aspect of fog computing in an industrial scenario concerns the orchestration of applications to be executed on an existing automation system network. This paper studies the problem of mapping distributed applications onto an industrial fog network so as to minimize data transfer cost while adhering to resource constraints. The resulting fog application allocation problem is NP-complete and can be described as an integer linear program. We present and evaluate three polynomial-time heuristics, which exploit the problem structure of a typical fog application and are shown to find high quality allocations efficiently. Through extensive simulations on practically relevant scenarios, we show that the best performing heuristic has performance comparable to the optimal solution across most tested problem instances. The experiments with a real industrial fog application show that the best performing heuristic has an optimality gap of only 3.6%.**

***Index Terms*—Fog Computing, Allocation, Automation Systems**

I. Introduction

The cloud computing paradigm has become the center of gravity of Today's computing ecosystem. Being able to quickly activate a number of machines on a large cluster many kilometres away with the click of a button is a very flexible and user-friendly way of handling computing resources. For instance, it reduces capital expense of small to medium size companies and allows them to scale their services gracefully and dynamically depending on the size of the customer base.

Nonetheless, deploying an application in the cloud is not always an optimal, or even feasible solution. There are many cases where a more local solution is preferable, in particular in an industrial context. For instance, if an application is required to respond within millisecons to sensed events, the data must be processed *locally* near the origin of the data; if necessary, remote cloud resources may still be used for further processing or longer term analytics. Many industrial automation and control systems require sense-process-act cycles that take no longer than a few milliseconds, e.g., when controlling a robot arm. Offloading such control logic to the cloud is not feasible with Today's Internet. Moreover, in some cases the data produced by sensors is of such high volume or velocity that it is simply infeasible to ship it to the cloud directly without any pre-processing on site first. Finally, some operators of industrial sites prefer to keep certain data on site due to privacy or confidentiality concerns.

Fog Computing [1] is a paradigm that was introduced in 2012 and intends to bring the computational convenience of cloud computing to the edge of the network. In particular, fog computing harnesses the resources of devices and components along the edge-to-cloud continuum to provide a computational layer that can host applications flexibly and dynamically. Fog Computing has applications in many areas and is especially suited for applications where a lot of data is generated by connected sensors and smart devices, e.g., in *Internet-of-Things*-scenarios [2]. In the literature, possible applications involve smart vehicles, smart cities, content delivery networks [2], as well as applications in the context of "Industry 4.0", which includes smart industrial sites like highly automated factories or power grid substations [3].

The reality today is that computing resources in industrial sites are typically configured and deployed in a mostly manual and static way. As a consequence, industrial automation systems are inflexible in the sense that deploying new applications or new versions of applications requires high manual effort.

This paper addresses one important step towards the vision of fully fog-empowered industrial automation, where all computing resources are interconnected and ready to host an unlimited range of applications dynamically and efficiently; and where applications can be deployed, allocated, and orchestrated in a highly automated fashion such that resource constraints and application-specific placement constraints are respected and data movement is minimized. In particular, this paper investigates the problem of how to allocate available computational resources to the different components of a fog application optimally, i.e., optimizing data transfer cost while not exceeding resource capacities.

The remainder of this paper is organized as follows. Section II introduces the formal model and problem definition. Section III presents our algorithms, followed by an experimental evaluation in Section IV. We discuss related work in Section V and conclude in Section VI.

II. Model

A. Assumptions and Definitions

We refer to the computational devices of an (automation) system that participates in the fog as *fog nodes*. We denote communication links between fog nodes, e.g. Ethernet links, as *fog links*. Fog nodes and links make up the *fog network*.

†This research was done when Yvonne-Anne was a Prinicpal Scientist at ABB Corporate Research, Switzerland.

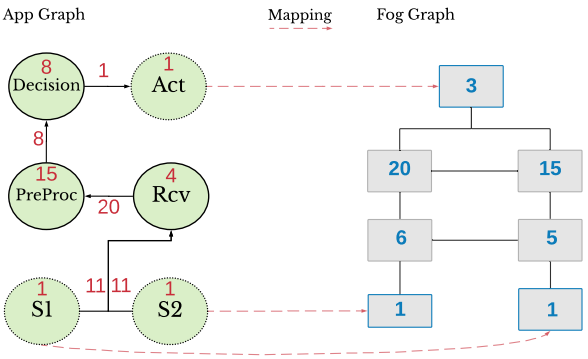


Fig. 1. Fog Application Allocation (FAA) problem example with three location-bound nodes ($S1$, $S2$, Act).

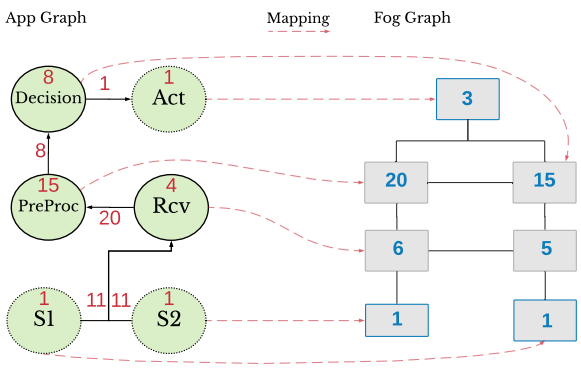


Fig. 2. Optimal mapping with cost $c = 11{\cdot}2{+}11{\cdot}1{+}20{\cdot}1{+}8{\cdot}1{+}1{\cdot}1 = 62$.

We assume that fog applications are comprised of computational components, so-called *app nodes*, which communicate with each other. The data communication between two app nodes is denoted by *app edge*. We assume that the fog may allocate different app nodes onto different fog nodes. Thereby the communication between app nodes is mapped to appropriate communication channels, e.g. to TCP/IP connections over Ethernet, if communication is across fog nodes, and to shared memory or virtual Ethernet, if both app nodes are allocated to the same fog node. Further assumptions are:

- *Resource Splitting*. Fog nodes have limited resources such as CPU or memory capacity. We assume that these resources can be split into arbitrary shares to be used by different applications.
- *Heterogeneity*. Fog networks are *heterogeneous*, i.e., they comprise various types of computing devices with different characteristics, such as different operating systems or different resource capacity levels. A fog network might include a switch or a controller with very limited processing capacity as well as a powerful multi-core server.
- *Reachability*. We assume that each fog node can be reached from any other fog node, wither via a direct link or via other intermediate fog nodes.
- *Shortest-path routing* We assume that communication between two fog nodes is routed along a shortest path.
- *Location-bound nodes*. Some app nodes are *location-bound*, i.e., they can only be allocated to a predetermined fog node, e.g. on a sensor or actuator.

B. Problem Definition

A *fog network* is modeled as a weighted graph $F = \hat{V}, \hat{E}, \hat{w})$. The set of fog nodes is denoted by $\hat{V}$. The set fog links is denoted by $\hat{E}$. There is an edge $e = (u, v) \in \hat{E}$ if and only if there is a direct communication link between u and v. Each fog node $v \in \hat{V}$ has a limited capacity, which is given by function $\hat{w}\colon \hat{V} \to \mathbb{R}$. The number of nodes and edges of the fog network is denoted by $\hat{n}$ and $\hat{m}$ respectively. A *fog application* is modeled as a weighted directed graph $A = (V, E, w, \omega)$. Set V denotes the app nodes. The app edges E $\subseteq V \times V$ represent the directed data flow among the app nodes. Weight function $\omega\colon E \to \mathbb{R}$ represents the communication cost of an app edge per hop. One natural interpretation of ω is the bandwidth required on a fog edge for handling the app edge communication. Each app node $v \in V$ has a weight given by function $w\colon V \to \mathbb{R}$, which represents the resource requirements of v, e.g. the needed CPU capacity or memory. [1] The number of nodes and edges of a fog application is denoted by n and m respectively. $L \subseteq V$ denotes the (possibly empty) set of location-bound app nodes. Their mapping is given by $\sigma\colon L \to \hat{V}$. An allocation function $\mu\colon V \to \hat{V}$ is called *feasible* if it maps all location-bound app nodes correctly, i.e., for all $v \in L$ it holds that $\mu(v) = \sigma(v)$, and capacities are not exceeded, i.e.,

$$\forall\ \hat{v} \in \hat{V} : \sum_{v \in \mu^{-1}(\hat{v})} w(v) \leq \hat{w}(\hat{V}),$$

where μ^{-1} is the inverse relation to μ, i.e., $\mu^{-1}(\hat{v}) = \{x \in V \mid \mu(x) = \hat{v}\}$. The *cost* c_μ of an allocation μ is given by the total communication cost among the allocated app nodes along shortest paths, i.e.,

$$c_\mu = \sum_{e=(u,v) \in E} \omega(e) \cdot d(\mu(u), \mu(v)),$$

where the distance function $d : \hat{V} \times \hat{V} \to \mathbb{N}_0$ is given by the minimum number of hops (fog links) between two fog nodes. Note that, thus, communication between collocated app nodes is free, since $d(\hat{v}, \hat{v}) = 0$ for any $\hat{v} \in \hat{V}$. Finally, we are ready to define the fog application allocation problem formally.

Definition 1 (FAA Problem)**.** *Given an application graph A, a fog graph F, the location-bound nodes L and their mapping σ, the* Fog Application Allocation (FAA) problem *is defined as finding a feasible allocation μ that minimizes cost c_μ.*

[1] Instead of a single value, the requirements can be represented by a vector with one entry for each of the different types of resources. For simplicity, we present the single value case in this paper, all our results can be extended to the more general case by normalizing the different dimensions and using vector norms.

Note that the effect of the optimization objective is that app nodes with costly communication among them are allocated close to each other. The objective thus avoids unnecessary data transfers over physical network links and at the same time reduces communication latencies. Both effects are desirable in industrial automation systems.

Further note that the model refrains from specifying a capacity on the fog links. The reason for this omission is that we consider this a minor simplification, as the optimization objective already is to keep the communication cost low; adding additional edge constraints would merely complicate the model. Furthermore, we consider a use-case where FAA instances are explored as part of engineering and sizing of the physical systems, i.e., network capacities are sized relative to the expected load.

C. Example

Figure 1 illustrates an example of the allocation problem with a simple problem instance. It shows an application graph on the left (green), the physical fog network on the right (grey) and the location-bound mapping illustrated by the dashed red arrows. The application graph has six nodes with numbers inside the vertices which represent the computational costs w of the app nodes. The numbers on the edges represent the used network bandwidth ω of the app edges. The fog nodes are labelled with their resource capacity $\hat{w}$. The application nodes $S1$, $S2$ and Act (dotted) are location-bound nodes which have to be mapped to their corresponding fog nodes (blue border). The application represents a control loop with two sensors ($S1, S2$) which send data to the receiver node (Rcv). The receiver node then forwards the data to a preprocessor ($PreProc$), which aggregates the data and sends it to a node that calculates a decision ($Decision$). The decision is forwarded to an actuator (Act), which executes the decision. Such control loops are often used in industry, e.g., in a manufacturing process where faulty items are detected by a camera (sensor) and removed from the assembly line.

An optimal allocation for the problem instance in Figure 1 is depicted in Figure 2.

D. Complexity

The FAA problem is NP-complete in general. Proofing NP-completeness can for instance be achieved by a reduction from the bin-packing problem. The question of whether a FAA instance is feasible entails the question of whether app nodes can be distributed onto fog nodes so that no node capacity is exceeded, which is a standard bin-packing problem. Refer also to [4] for NP-completeness and inapproximability proofs for a family of closely related allocation problems with additional constraints.

While the problem is defined here for general application and fog graphs, we strive to find heuristics that yield good results for typical industrial setups. As detailed in Section IV we identified series-parallel graphs and cactus graphs as suitable graph classes to represent a large part of industrial use-cases. Moreover, the order of both types of graphs is typically expected to be in the 100s.

Minimize

$$\sum_{(u,v)\in E} \left(\omega(u,v) \cdot \sum_{\hat{u},\hat{v}\in \hat{V}} d(\hat{u},\hat{v}) \cdot z(u,v,\hat{u},\hat{v}) \right)$$

subject to

$$\sum_{\hat{u}\in\hat{V}} y(u,\hat{u}) = 1 \qquad \forall u \in V \quad (1)$$

$$\sum_{u\ \in\ V} w(u) \cdot y(u,\hat{u}) \leq \hat{w}(\hat{u}) \qquad \forall \hat{u} \in \hat{V} \quad (2)$$

$$y(u,\hat{u})=1 \qquad \forall (u,\hat{u}) \in L \quad (3)$$

$$z(u,v,\hat{u},\hat{v}) \leq y(u,\hat{u}) \quad (4)$$

$$z(u,v,\hat{u},\hat{v}) \leq y(v,\hat{v}) \quad (5)$$

$$z(u,v,\hat{u},\hat{v}) \geq y(u,\hat{u}) + y(v,\hat{v}) - 1 \quad (6)$$

$$\forall (u,v) \in E, \hat{u},\hat{v} \in \hat{V} \quad (4\text{-}6)$$

Fig. 3. FAA formulation for ILP solvers.

III. Allocation Algorithms

In this section, we present different algorithms for solving FAA. We show how to solve FAA optimally with integer linear programming (ILP) techniques and propose three polynomial-time heuristics.

A. Optimal Allocator

The FAA problem definition in Section II essentially defines an integer linear program. By applying some modifications, we receive the ILP formulation in Figure 3, which can be solved by common ILP solvers for small enough instances.

Instead of the mapping function μ, we introduce a binary function $y : (V \times \hat{V}) \rightarrow \{0,1\}$ that indicates whether an application node is mapped to a fog node or not. To indicate how an application edge is mapped to a shortest path between fog nodes we introduce $z : (E \times \hat{V} \times \hat{V}) \rightarrow \{0,1\}$. It holds that $z((u,v),\hat{u},\hat{v}) = 1$, if app node u is mapped to fog node $\hat{u}$ and v is mapped to fog node $\hat{v}$. Function z serves the purpose of avoiding a nonlinearity in the cost function ($y(u,\hat{u}) \cdot y(v,\hat{v}) = z(u,v,\hat{u},\hat{v})$).

The first constraint (1) ensures that each app node is mapped to exactly one fog node. The second constraint (2) ensures that the fog node capacities are not exceeded by the mapped app nodes. The third constraint (3) ensures that all location-bound app nodes are mapped to their assigned fog node. The last three constraints (4-6) connect the binary decision function y with the helper function z.

B. Allocation Heuristics

Since FAA is NP-hard in general, it is infeasible to solve instances above a certain scale and complexity optimally. As a consequence, we present three heuristics for FAA in the following. The heuristics are tailored to the communication cost minimization objective in varying degrees.

Although the three presented heuristics vary in their level of sophistication, they all share some common features. They all follow a greedy approach and find mappings for the app nodes

Algorithm 1 Node Constraints Allocation

```
1: procedure NCA(AppNodes, FogNodes, LbNodes)
2:     sAppNodes ← SortByWeight(AppNodes)
3:     sFogNodes ← SortByWeight(FogNodes)
4:     μ ← new Mapping
5:     AddAllLocationBound(μ, LbNodes)
6:     for all a ∈ sAppNodes \ LbNodes do
7:         f = FindFirstFit(sFogNodes, a)
8:         if IsDefined(f) then
9:             AddMapping(a, f)
10:        else return 0
11:    return μ
```

Algorithm 2 Greedy Collocation Allocation

```
1: procedure GCA(AppGraph, FogNodes, LbNodes)
2:     sEdges ← SortByWeight(AppGraph.edges)
3:     μ ← new Mapping
4:     AddAllLocationBound(μ, LbNodes)
5:     for all (a1, a2) ∈ sEdges do
6:         (f1, f2) ← BestFogNode(a1, a2)
7:         if IsDefined(f2) then
8:             AddMapping(a1, f1)
9:             AddMapping(a2, f2)
10:    for all a ∈ UnmappedNodes(μ, AppGraph) do
11:        f ← FeasibleFogNode(a)
12:        if IsDefined(f) then
13:            AddMapping(a, f)
14:        else return 0
15:    return μ

16: procedure BestFogNode(a1, a2)
17:    if a1, a2 ∈ μ then return 0
18:    if a1, a2 ∉ μ then
19:        return FeasibleFogNode(a1, a2)
20:    return ClosestToMapped(a1, a2)
```

iteratively. They all may deem a problem instance infeasible, even though a feasible solution exists.

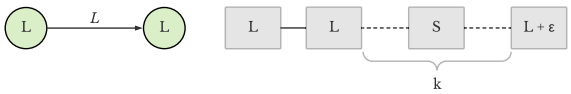
1) Node Constraints Allocation (NCA): The NCA heuristic is the most straightforward of the three presented heuristics and serves as a baseline in our evaluations. NCA focuses on finding a feasible solution regarding the resource constraints of the fog nodes and thereby ignores the communication cost optimization objective. As the constraints are not dependent on the connections i.e. edges between the nodes, the problem that NCA solves, is a bin-packing problem.

Refer Algorithm 1 for a description of NCA in pseudo code. As a first step, the app nodes and fog nodes are sorted by their weight in descending order. In line 4, the mapping μ is initialized. In a next step, all location-bound nodes are added to the mapping (AddAllLocationBound, line 5). Subsequently, the remaining app nodes are mapped in order of their weight. Thereby the sorted fog node list is traversed until a fog node with enough remaining capacity is found in a first-fit manner (FindFirstFit). If no such fog node is found, i.e., no fog node has enough capacity left to host the app node, the problem instance is deemed infeasible and return 0. After all nodes are mapped, the algorithm returns the completed mapping.

We chose NCA to implement a first-fit greedy bin-packing, since the first-fit approach likely collocates several app nodes onto one fog node. NCA thus implicitly reduces the cost, as communication between collocated app nodes results in zero cost. We also expect this algorithm to perform well in finding a solution, but to produce results with suboptimal costs for many problem instances, as it strives for feasibility and not for optimality.

Time Complexity. The running time of NCA is in $\mathcal{O}(n \log n + \hat{n} \log \hat{n} + n \log \hat{n}) = \mathcal{O}(N \log N)$, where $N = \max\{n, \hat{n}\}$ is the maximum of the number of app nodes and the number of fog nodes. First, the list of app nodes as well as the list of fog nodes are sorted, which takes $\mathcal{O}(N \log N)$. Second, NCA iterates over all app nodes calling FindFirstFit, with a time complexity of $\mathcal{O}(n \log \hat{n})$.

While a first-fit approach finds a 2-approximation for the bin-packing problem (with respect to minimal number of used bins), NCA can lead to arbitrarily bad allocations regarding our cost function, even when both the fog network and the app graph are chains and the maximum weight L is a constant factor more than the minimum weight S. The following example shows this shortcoming of NCA:



The app graph connects two heavy nodes (weight L) with a heavy edge (weight L). The fog graph is a chain (or line) graph which consists of two nodes of weight L followed by $k-1$ nodes of small weight S and ends with a node that has a slightly higher capacity than L, namely $(L+\varepsilon)$.

In an optimal allocation, the app nodes are mapped to the first two neighboring L fog nodes, yielding a cost $c_{OPT} = L$. In contrast, NCA maps the first app node to the largest fog node $(L+\varepsilon)$ and the second to one of the remaining large fog node resulting in a cost of $c_{NCA} = kL$. Thus, the cost of the solution computed by NCA can exceed the optimal allocation cost by a factor in the order of $\Omega(\hat{n})$.

Note that for all three presented heuristics, there exist feasible problem instances for which the algorithms fail to find a feasible allocation. Thus, for all approximation quality investigations, we consider only instances for which the algorithm finds a feasible solution. Bounds on the approximation quality therefore give hints as to how close or far from optimal a found allocation can be.

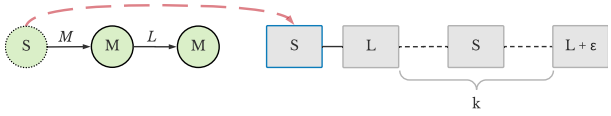
2) Greedy Collocation Allocation (GCA): The second heuristic, GCA, tries to avoid high communication cost by collocating neighboring app nodes with heavy edges in between on the same fog node, or allocating them as close from each other as possible. In particular, GCA greedily processes the edges from the heaviest to the lightest, and tries to map unmapped incident app nodes as close to each other as possible.

See Algorithm 2 for a description of GCA in pseudo code.

In a first step, a list of all app edges descendingly ordered by their weight is created (line 2). After adding the location-bound nodes to the mapping (line 4), the main loop of the algorithm iterates over the sorted edge list and assigns the incident app nodes with BESTFOGNODE(line 6). Ideally, the same fog can be used for both app nodes such that the edge weight does not occur in the cost. But as this is not always possible, BESTFOGNODE addresses three different cases: if both app nodes are already mapped, it returns nothing (line 17). If both nodes are unmapped, FEASIBLEFOGNODE is called, which returns the fog node with most remaining resources, if its capacity suffices to host both nodes, otherwise it returns zero, i.e., mapping the app nodes is postponed. If one node of the app nodes is already mapped, BESTFOGNODE maps the other app node as close as possible to the first one (line 20). After the algorithm has finished iterating over the edges, it is possible that there are some remaining unmapped app nodes. These nodes either did not fit in the previous loop or their degree is zero. The algorithm places them on the fog nodes with highest and sufficient remaining capacity, with FEASIBLEFOGNODE (lines 11-13). If no such node can be found, the instance is deemed infeasible (line 14).

Time Complexity. GCA runs in $\mathcal{O}(m \log m + m\hat{n} + n \log \hat{n}) = \mathcal{O}(m\hat{n}) = \mathcal{O}(N^3)$. First the app edges have to be sorted ($\mathcal{O}(m \log m)$), followed by two loops of which one iterates over all app edges and the other one over all fog nodes. The first loop calls BESTFOGNODE which involves traversing the fog graph in order to find the closest fog node with enough capacity left, which takes $\mathcal{O}(\hat{n})$. The call FEASIBLEFOGNODE can be implemented in $\mathcal{O}(\log \hat{n})$, which also holds for CLOSESTTOMAPPED .

As GCA does not exploit knowledge on location-bound nodes or other already allocated nodes, we can construct chain graphs incurring a high cost compared to the optimal solution. Depending on the scenario, GCA places nodes very far from the area of the mapped location-bound node, which results in long communication paths. The following figure illustrates such a case, where $L = 2M = 4S$ for the nodes and and edges. The application graph contains a location-bound node S which is mapped on the fog node S (blue border).



In an optimal mapping the M-app nodes are mapped closely to the location-bound node and thereby minimize the cost. GCA collocates the M-app nodes on the fog node with the most remaining resources without taking pre-allocated nodes into account and the cost can be in the order of $\Omega(k) = \Omega(\hat{n})$ time larger than the optimal solution.

3) Greedy Border Allocation (GBA): The Greedy Border Allocation approach maps app nodes as close to their already mapped neighbors as possible using a weighted breadth-first traversal of the app graph starting from the location-bound nodes. Thereby, the algorithm keeps track of the *border* edges, i.e., edges where one incident app node is mapped and the other is not. GBA greedily processes the heaviest border edge and updates the border, until all app nodes are mapped.

Algorithm 3 Greedy Border Allocation

```
1:  procedure GBA(AppGraph, FogNodes, LbNodes)
2:      μ ← new Mapping
3:      AddAllLocationBound(μ, LbNodes)
4:      sBE ← SortedBorderEdges(AppGraph, μ)
5:      while IsIncomplete(μ) do
6:          (a1, a2) ← NextEdge(μ, sBE)
7:          f ← ClosestFeasibleFogNode(a2, μ(a1))
8:          if IsDefined(f) then
9:              AddMapping(a2, f)
10:             UpdateBorderEdges(a2)
11:         else return 0
12:     return μ

13: procedure NextEdge(μ, sBE)
14:     (a1, a2) ← LargestEdge(sBE)
15:     if (IsDefined(μ(a1)) then return (a1, a2)
16:     (a1, f) ← NextFromDisjointPart(μ)
17:     if IsDefined(f) then
18:         AddMapping(a1, f)
19:         UpdateBorderEdges(a1)
20:         (a1, a2) ← LargestEdge(μ, sBE)
21:         if IsUndefined(b1) then goto 16
22:         return b1, b2
23:     else return 0
```

This way of traversing the graph solves the problem we encountered with GCA. The algorithm has two advantages: on the one hand, it implicitly reduces the node weights by clustering the app graph densely in one area of the fog graph and on the other hand, it uses the edges of the location-bound nodes first. If we assume that the location-bound nodes often output a large fraction of the totally exchanged data (e.g. a periodically probing sensor), the algorithm is able to minimize the contribution of those edges to the cost function first.

Algorithm 3 describes GBA in pseudo code. After mapping all location-bound nodes, GBA creates the set of all edges incident to an already mapped node (SORTEDBORDEREDGES), sorted by their weights in decreasing order. After this preparation step, the next app node to map is determined until no nodes are left. To this end, NEXTEDGE retrieves the largest edge from the current edge list (*sBE*). If there are no edges in this list, this means that the app graph is unconnected or that there are no location-bound nodes. In this case, the algorithm determines the app nodes which are disjoint to the already mapped and chooses an arbitrary one (line 19). If there is none, the algorithm terminates. If there is one, the edges adjacent to this app node are added to the border edges and the largest edge is returned (lines 17-22).

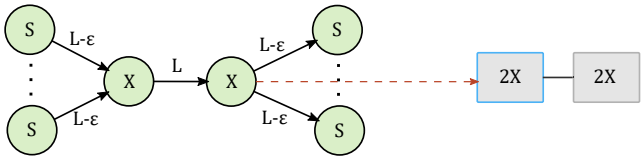
With NEXTEDGE returning the edge with the already mapped app node $a1$ and the next app node to map $a2$ (line 6) the algorithm proceeds as follows. It chooses the fog node closest to the already mapped $a1$ with the call to CLOSESTFEASIBLEFOGNODE. This means that it chooses the same fog node if there are enough resources left or the closest

with enough resources left in terms of hops determined by a breadth-first traversal. Having found the fog node, it adds it to the mapping and adds all adjacent edges of $a2$ to the *sBE* using UPDATEBORDEREDGES. The call UPDATEBORDEREDGES also removes edges whose app nodes are both already mapped (lines 8-12). If no fog node with enough remaining capacity is found, the algorithm deems the instance infeasible.

Time Complexity. GBA has a time complexity of $\mathcal{O}(m \log m + m(n + \log \hat{n} + \hat{n}) + m) = \mathcal{O}(m \log m + mn + m \log \hat{n} + m\hat{n}) = \mathcal{O}(N^3)$. In a first step, the app edges are sorted and can be stored as a max-heap in $\mathcal{O}(m \log(m))$. The main loop iterates over all app edges in *sBE*, i.e. in $\mathcal{O}(m)$. One iteration of the main loop is broken down as follows:

- The procedure NEXTEDGE has a complexity of $\mathcal{O}(n + \log \hat{n})$ as it calls following procedures:
  - LARGESTEDGE: retrieves the maximum element in a max-heap in constant time, $\mathcal{O}(1)$.
  - NEXTFROMDISJOINTPART: assuming a heap data structure, finding the disjoint part as well as finding a fog node with enough capacity takes $\mathcal{O}(n + \log \hat{n})$.
  - ADDMAPPING: constant time hash table update.
  - UPDATEBORDEREDGES: takes $\mathcal{O}(\log n)$, as the additional edges have to be inserted into the edge heap.
- CLOSESTFEASIBLEFOGNODE: takes $\mathcal{O}(\hat{n})$ to search fog nodes.
- UPDATEBORDEREDGES: takes $\mathcal{O}(m)$ in the worst case since up to a constant fraction of all app edges are concerned.. However, since each edge is only added or removed once in the whole GBA, the overall time spent with this procedure is also in $\mathcal{O}(m)$.

In general, GBA also does not always find good allocations. Consider a fog app graph with two connected nodes of weight X; each of those nodes has k additional neighbors of weight S. Let the edge weight between the two X-nodes be L, and let all other edges have weight $L - \varepsilon$. Furthermore, let $kS \leq X$. Finally, let the fog graph consist of two nodes with capacity $2X$, and let one X-node be bound to one fog node:



An optimal mapping collocates each X-node together with its S-neighbors, which yields a cost of $c_{opt} = L$, since only the communication between the two X-nodes goes across one fog link. By contrast, GBA maps the second X-node to the same fog node as the location-bound X-node before it considers any S-node. This is since the X-nodes are connected by the heaviest border edge. Afterwards, one fog node has no capacity left, thus all S-nodes are allocated to the other fog node. The resulting cost is $c_{GBA} = 2k \cdot (L - \varepsilon)$. Thus, a feasible solution computed by GBA can exceed the optimal allocation cost by a factor in $\Omega(n)$. This example can also be used to show that $c_{GBA} \in \Omega(n)$.

Allocator	Time complexity	Approx. lower bound
NCA	$\mathcal{O}(N \log N)$	$\Omega(\hat{n})$
GCA	$\mathcal{O}(N^3)$	$\Omega(\hat{n} + n)$
GBA	$\mathcal{O}(N^3)$	$\Omega(n)$

TABLE I
OVERVIEW OF ALLOCATION HEURISTICS PERFORMANCE IN TERMS OF RUNNING TIME AND LOWER BOUNDS OF THE APPROXIMATION RATIO. N DENOTES THE MAXIMUM NUMBER OF APP OR FOG NODES.

4) Heuristics comparison: As we have seen, although the presented heuristics are all of greedy nature, they have different advantages and disadvantages. The performance of all three can be worse than the optimal up to a factor linear in the number of app or fog nodes. See Table I for an overview. Yet, as we will see later, they perform well on practically relevant FAA problem instances.

5) Combination: To improve the performance of our allocators without a significant growth in running time, the three heuristics can be combined as follows. First, execute all three heuristics on a problem instance and then choose the best result. Second, randomize the selection of equivalent next edges or nodes to process or map and run the heuristics multiple times. This will lead to a more diverse set of solutions and improved results.

IV. EVALUATION

We evaluate our allocation algorithms with respect to quality, running time and feasibility ratio. To this end, we built a simulator to generate different kinds of problem instances, plug-in new allocators and even to modify the cost function with a few lines of code. This allows us to run various experiments and adapt them according to our needs. We have published the simulation framework as open source software under the name *FAAP Simulator*[2].

A. Experimental Setup

Fog Network Topologies. Communication networks of automation systems typically follow a hierarchical tree structure, of which some parts form rings for redundancy purposes. Thus, cactus graphs[3] can model many industrial fog networks. Furthermore, it has been shown that 32% of all network topologies of well-known internet topology-datasets belong to the family of cactus graphs [5]. Consequently, we use cactus graphs for the underlying physical network in our experiments.

Fog App Graph. We decided to use series-parallel type of graphs for modeling realistic application graphs, since they are known to widely capture common data streaming tasks. For instance, they cover all map-reduce applications. We use the Series-Parallel Decomposable (SPD) graphs as defined in [6]. The recursive definition of SPD graphs allows for a straightforward algorithm to generate a random SPD graph of given order.

Since applications in automation systems often have location-bound nodes, we define a fixed ratio of all app

[2]Available at https://github.com/MSuter6/faap-simulator

[3]The class of cactus graphs comprises all graphs where each edge is part of at most one simple cycle.

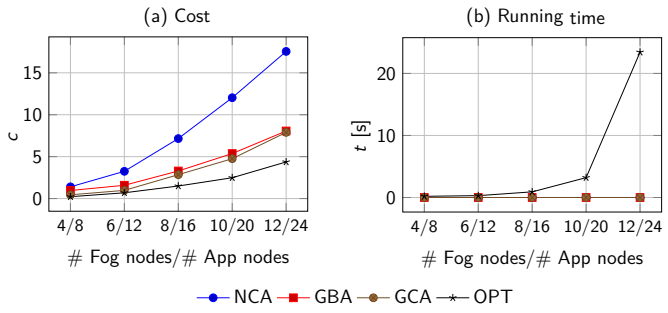


Fig. 4. Cost and running time performance for small instances.

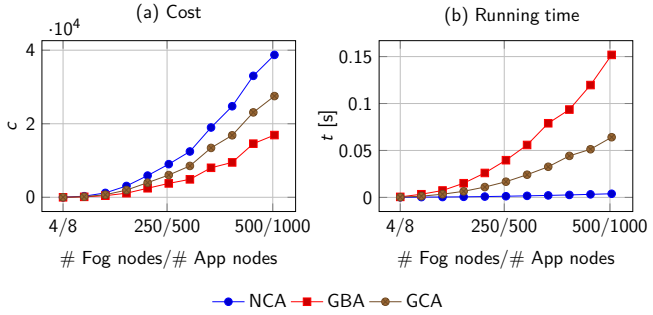


Fig. 5. Cost and running time performance for larger instances.

nodes to be location-bound. We select an assignment of these nodes to Fog Network nodes uniformly at random under consideration of the capacities.

We run each experiment 100 times with randomly generated cactus graphs for Fog graphs and SPD graphs for app graphs. For most experiments, we chose an order ratio $n/\hat{n} = 2$, because we expect to run large or multiple applications with more app nodes than fog nodes on our fog network. We ran experiments up to a scale of 500 fog nodes and 1000 app nodes, which is the scale we expect industrial fog networks and applications to operate within. Values for fog and app node weights are chosen uniformly at random in (0,1) and (0,1/3) respectively. Hence the average fog node capacity of a single fog node is three times larger than the average resource requirement of a single application node. Since the values can vary widely for different nodes this models the heterogeneity of the applications and the devices. We used Gurobi[4] to solve the ILP and we ran our experiments on a machine with a quad core CPU with a clock rate of 3.3 GHz and 24 GB of RAM.

B. Results on Synthetic Graphs

In a first experiment series, we examine the impact of the number of nodes. Since the time needed to solving ILP grows exponentially with the number of nodes, we compared the heuristics with the ILP solver for small scale instances only.

Figure 4 depicts the average cost and running time of the heuristics and the ILP solver. The running time of the ILP solver is orders of magnitude larger than the heuristics, thus we use a logarithmic scale in Figure 4(b). As expected, the running time of the solver increases exponentially with growing problem scale. The hardest instance takes the solver about 500s to solve (not visible in the plot as only averages are shown there to depict trends). The heuristics have a consistent performance with a low variance in terms of allocation time which is within the microseconds range. Thereby, the NCA takes slightly less time than the GCA, which in turn takes slightly less time than the GBA.

As expected the cost of the allocations computed by the heuristics exceed the optimal cost. In the evaluated range, the performance ratio reaches a factor of 3 for NCA, while GCA and GBA accrue up to 1.5 of the cost on average.

[4]Commercial MILP solver, http://www.gurobi.com

When evaluating the same metrics for the heuristics on a larger scale (see Figure 5), we observe that NCA is significantly faster than the other allocators. All running times are in the sub-second range, and thus near real-time recomputation of the allocation with changing instances should not cause any problems for graphs in the expected size range.

The feasibility rate, i.e., the number of instances for which an allocator finds a feasible solution divided by the number of feasible instances, varies between the different allocators. As expected, NCA has the best performance with respect to this measure and finds as many feasible instances as the optimal allocator. GCA shows the worst performance in this regard, while GBA is slightly better.

Furthermore, we analyzed the behavior of the algorithms when varying the following properties of the instances: ratio of app versus fog graph size, density, fog node capacity, app node requirements, number of location-bound nodes. Overall, we observe a consistent behavior of the three heuristics to the results mentioned above, also when considering percentiles and direct comparisons to the optimal solution. We omit the plots of these experiments due to lack of space (the simulation framework on github includes the scripts to generate them).

C. Industrial Use Case

To complement our experiments on purely synthetic graphs, we present now a concrete fog allocation use case with a real fog application and a realistically modeled fog network.

Control Application We use a control application that is currently deployed on various industrial sites as an exemplary fog application. As the details of the application are confidential, we call this application the *ABB Control Application* and use an anonymized notation. Nevertheless, the application provided by ABB comes with a detailed structure as well as weights for the node and edges. The weights were gathered from a deployed application of this type.

Figure 6 illustrates the structure of the ABB Control Application including its node and edge weights. The application itself consists of a control loops for each of K subsystems containing a sensor (S_i) and an actuator (T_i) as well as a preprocessing block (A_i–E_i). The app graph order n is dependent on K: $n = 7K + 3$. The output of all control loops is aggregated in a global control state (H) and then

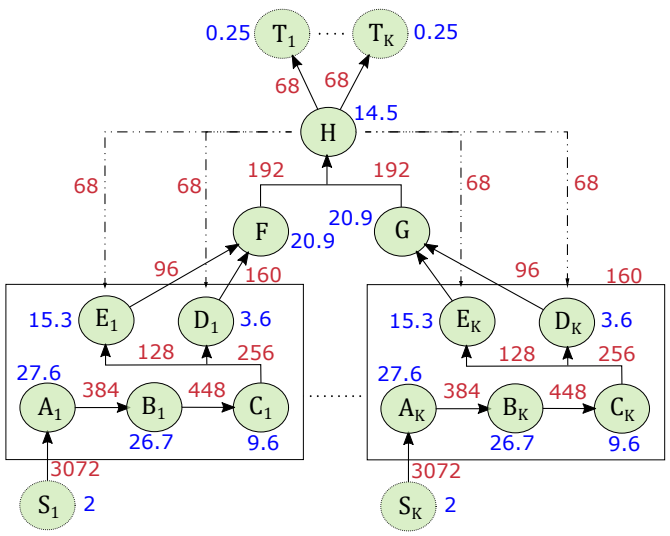


Fig. 6. Scheme of the ABB Control Application. Node weights in MIPS (blue), edge cost in kbit/s (red).

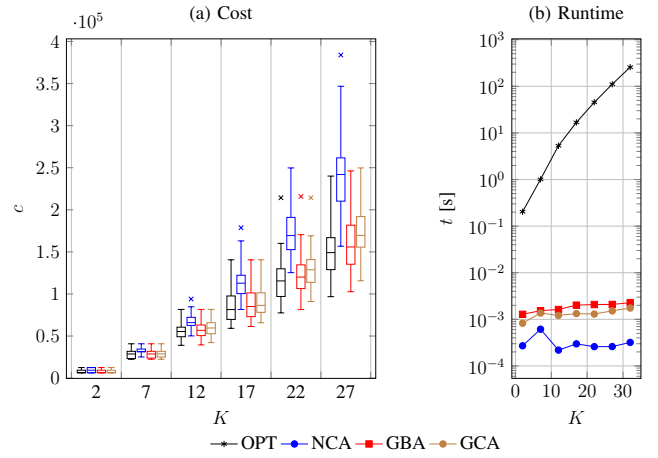


Fig. 7. Performance on ABB Control App with optimally solvable instances.

fed back to the actuators, which are collocated on the same fog node as the corresponding sensor. There exist two nodes, which act as partial aggregators (F,G) and receive the output from preprocessing blocks. Both partial aggregators receive a balanced amount of data. F receives data from preprocessing blocks with $i \leq K/2$ and G data where $i > K/2$.

The resource demand of the app nodes ranges from 0.25 Million Instructions per Second (MIPS) for the actuators to 27.55 MIPS and for the edges from 68 kbit/s to 3072 kbit/s for the sensor output. The high sensor output, which is about a factor 10 larger than the second largest edge, supports our assumption (Section 3) that location-bound nodes tend to produce a large fraction of the exchanged data. The ratio of the number of location-bound nodes to all fog nodes is $2K/(7K+3) < 0.3$.

Fog Network. To represent typical automation systems of industrial sites such as factories or substations, we assume the following resource capacity mix:

1) *Resource constraint IoT devices* make up about *60%* of all fog devices. They have a resource capacity range between *50-200 MIPS*.
2) *Medium-sized devices*, e.g., controllers, cover around *20%* of all fog devices with a resource capacity ranging between *500-2000 MIPS*.
3) *Powerful devices*, e.g multi-core servers, represent about *20%* of all fog devices. They have a resource capacity range between *4000-10000 MIPS*.

To account for operating system and other, non-fog software already running on the machines, 25% of resources are assumed to be available for fog applications.

For the network topology, cactus graphs are generated such that K nodes are available for the location-bound application nodes and the additional nodes offer enough capacity to host the ABB Control Application in expectation using the above assumptions on resource availability. In particular, the order $\hat{n}$ of the fog network is chosen as

$$\hat{n} = \left\lceil \frac{\sum_{v \in V} w(v)}{\bar{v}} \cdot 4 \right\rceil + K = \left\lceil \frac{84.9K + 56.25}{862.5} \cdot 4 \right\rceil + K.$$

D. Results on Industrial Case Study

All box plots in this section show the median, 25th and 75th percentile (box) with whiskers at the closest value within 1.5 times the box (fence). Values outside of the fences are shown as outliers (x). Figure 7 shows the cost for K between 2 and 27 in steps of 5. Interestingly, it is feasible to use the optimal solver for much larger problem instances of this application than in the more generic synthetic experiments. With K = 27, we use 192 app nodes and 11 fog nodes, i.e. a large ratio between the fog graph order and app graph order. The fog graph order has a larger impact on the ILP formulation than the app graph order. This is due to the quadratic number of fog node pairs which are included in the ILP formulation for listing all shortest paths.

While the value of K can become significantly larger than 30 for some deployments, K is between 10 and 30 in a majority of cases in practice. Thus, it is a reasonable option to compute the allocation optimally in many deployments of the studied application, in particular, if the allocation is rather static and is not required to run frequently. In more dynamic environments, a fast heuristic or a hybrid solution, which resorts to the approximation provided by a fast heuristic after a given timeout, should be the preferred choice.

Compared to experiments with synthetic instances the heuristics exhibit a significantly improved cost performance. A look at the performance ratio $r_c = c_{OPT}/c_{ALG}$ shows an increased performance and a decreased variability. For instance, with $K = 27$, the GBA has a ratio of 0.964 which means that it only performs 3.6% worse than the optimal solution. The other heuristics have a ratio of $r_{NCA} = 0.668$ and $r_{GCA} = 0.868$. For the GBA, the relative deviation from the optimal solution stays nearly constant, while for the other

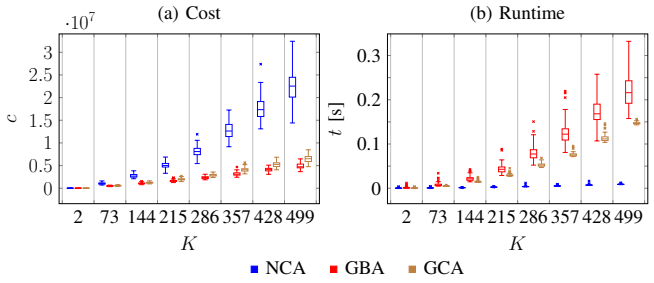


Fig. 8. Performance on ABB Control App on large instances.

two heuristics it gets slightly worse with growing K. We observe an exponential increase for the running times of the solver while the running times of heuristics stay nearly linear in the evaluated range.

1) Large Scale Instances: Figure 8 (a) shows the evaluation with K ranging from 2 to 499 in steps of 71. The largest instances comprises 3496 app nodes and 696 fog nodes. As problems of this scale are intractable for the solver, we only use the heuristics in the evaluation.

Similar to the results of the previous experiments, GBA performs best overall. GCA performs slightly worse on all instances. Both GBA and GCA exhibit a very low variability. NCA performs worst of all three, diverging more and more with respect to the costs for growing K.

In addition to the plotted data, we have measured the *feasibility rates*, i.e. the percentage of instances for which a given allocator found a feasible solution. All heuristics exhibit very similar feasibility rates, which are within 1% of the optimal. The feasibility rate is 65% at $K = 2$, 80% at $K = 17$, 92% at $K = 73$, 97% at $K = 144$, and reaches 100% at $K \geq 357$.

E. Discussion

The experiments on general performance of the allocators with different problem instance configurations reveal some interesting insights. We present a short recapitulation of the most relevant results:

a) Instance complexity: We found the graph order to be the parameter which is the main contributor to the scale and complexity of a FAA instance. We were able to optimally solve instances of up to 12 fog nodes and 24 app nodes using the solver. With the heuristics, instances with up to 500 fog nodes and 1000 app nodes are solved within about 100-200 ms.

b) Optimal solution: Our experiments show that an ILP solver is only applicable for small instance scales, with running times of 1 to 20 seconds. With growing instances, the running time increases exponentially and a solution becomes computationally intractable.

c) Heuristics performance: Overall, we can conclude that GBA performs best, especially if the instances are of larger scale. For smaller instances, GCA may outperform GBA, especially if they contain many location-bound nodes. In terms of feasibility rate, NCA has the best performance, but leads to significantly higher costs than the GCA and GBA for all instance types evaluated. The running time of the heuristics is in the range of milliseconds even for large instances, which allows for near real-time allocation. The NCA is the fastest of all allocators, followed by GCA. GBA is the slowest.

d) Performance on real-world example: For the ABB Use Case, we found the heuristics to perform even better than for more general examples. For the compared instances with $K \leq 27$, we found that the cost of GBA is only about 3.5% higher than the cost of the optimal solution, while the running time is exponentially faster.

V. Related Work

Fog Computing Due to the rather recent inception of fog computing, most papers on the topic, among them [1]–[3] describe the concepts of fog computing in general or describe allocation only on an abstract level, e.g. [7], [8]. A survey of different fog application models and application allocation algorithms is presented in [9]. It lists a multitude of different approaches in a systematic way, based on the constraints, objectives and method applied. Among work that considers the network topology as well as application dependencies, [10] exploits mobility while [11] explores trade-offs arising with varying user demand frequencies. Both aspects are irrelevant for industrial applications and the requirements are more stringent. [12] addresses application provisioning in fog computing-enabled IoT. It introduces a formal allocation model, but focuses on applications of a specific structure. We aim to apply to more general cases and propose an objective function tailored to automation systems. In [13] a general stream processing placement problem for fog computing is presented. They formulate the problem as an integer linear program and investigate the performance of a set of heuristics compared to an optimal solution. Our formulation is more focused on the aspects important in automation settings and thus easier to solve. A very similar problem formulation to the one studied in this paper can be found in [14]. The authors propose and evaluate a heuristics which takes path length, resource scarcity and latency into account. Another approach is presented in [15] where the focus is on the latency of a deployment.

Virtual Machine Placement Studies of the problem of placing virtual machines on physical machines under certain objectives such as resource usage minimization or utility maximization typically focus on data centers, e.g., [16]. The application structure, the underlying networks as well as the applicable constraints and objectives differ, thus our work addresses a related but different problem class.

Virtual Network Embedding The Virtual Network Embedding (VNE) problem denotes the problem of mapping an application onto the physical network under certain objectives and constraints. The application and the physical network are modelled as a graph and the mapping includes the mapping of the nodes as well as the edges. Hence, the body of VNE results is closely related to the problems studies in this paper. There exist many different variants of the VNE problem, which differ in characteristics such as constraints (node constraints,

edge constraints, node restrictions), objective functions (energy minimization, utilization maximization, path length minimization), whether multiple apps should be mapped, whether the problem is online or offline, and whether communication paths can be split. Even the simplest formulations of VNE are NP-hard [17], many of them even in-approximable [4]. Therefore, besides MILP formulations, many heuristics have been developed. This includes greedy heuristics, relaxations of the problem as well as metaheuristics including ant colony optimization, particle swarm optimization, neural networks, genetic algorithms and simulated annealing [17].

The combination of constraints and requirements for the allocation of distributed applications in automation systems as described in this paper has not been addressed in VNE literature to the best of our knowledge. Another difference is VNE's focus on the path mapping problem. In automation systems, node mapping is more important, as the communication paths can typically not be chosen by fog applications and we strive for minimal model and problem complexity.

Service Function Chaining Service function chaining (SFC) denotes the mapping of sequences (chains) of service functions (SF) onto a physical network. In SFC, one of main difference to fog application allocation is the structure of the application graph. While a fog application can be a complex network of different app nodes and app edges, a service function chain consists of a set of network functions, which is traversed in a well defined order [18]. In other words, a fog application can be any directed graph, while a service function chain is restricted to be a directed *line graph*. Naturally this limits the applicability of SFC. [19] relaxes the line graph limitation to general graphs, but addresses still a fairly different problem.

VI. Conclusion and Future Work

We addressed the problem of mapping distributed applications for automation systems onto an industrial fog network with the goal to select optimal locations for the different part of the applications. The objective function penalizes sending high-bandwidth data along multiple hops and thus incentivizes hosting neighboring fog application nodes as close to each other as possible As the general problem is NP-complete, we devised three efficient heuristics which exploit the problem structure of typical fog applications. Our experimental evaluation shows that the Greedy Border Allocation heuristic has a comparable performance to the optimal solution for the majority of the tested problem instances. In particular, when running the heuristic on an existing real-world application class, an optimality gap of only 3.6% remains. As a consequence, we have implemented GBA as part of a Kubernetes-based fog framework, and successfully applied it to find allocations for distributed automation system applications.

There are several directions for future work. One direction is to devise and evaluate further heuristics with a higher level of sophistication or customization to problem subclasses. Furthermore, one could extend the FAA model, e.g., with deadlines on certain series of distributed tasks and data transmissions, or including hard bandwidth capacities on the fog node links. Finally, our model studies a one-shot problem. It would be worthwhile to study more dynamic settings where applications are added or removed over time, which can lead to a slow degrading of allocation quality.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its Role in the Internet of Things," in *1st MCC workshop on Mobile Cloud Computing*. ACM, 2012, pp. 13–16.

[2] S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues," in *Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[3] I. Stojmenovic, "Fog Computing: A Cloud to the Ground Support for Smart Things and Machine-To-Machine Networks," in *Australasian Telecommunication Networks and Applications Conf. (ATNAC)*. IEEE, 2014, pp. 117–122.

[4] M. Rost and S. Schmid, "Charting the Complexity Landscape of Virtual Network Embeddings," in *IFIP Networking*, 2018.

[5] Y.-A. Pignolet, S. Schmid, and G. Tredan, "Tomographic Node Placement Strategies and the Impact of the Routing Model," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, p. 42, 2017.

[6] R. Eidenbenz and T. Locher, "Task Allocation for Distributed Stream Processing," in *35th Conf. on Computer Communications (INFOCOM)*. IEEE, 2016, pp. 1–9.

[7] Y. Jiang, Z. Huang, and D. H. Tsang, "Challenges and Solutions in Fog Computing Orchestration," *IEEE Network*, vol. 32, no. 3, pp. 122–129, 2018.

[8] S. Hoque, M. S. de Brito, A. Willner, O. Keil, and T. Magedanz, "Towards Container Orchestration in Fog Computing Infrastructures," in *41st Computer Software and Applications Conf. (COMPSAC)*, vol. 2. IEEE, 2017, pp. 294–299.

[9] A. Brogi, S. Forti, C. Guerrero, and I. Lera, "How to place your apps in the fog-state of the art and open challenges," *arXiv preprint arXiv:1901.05717*, 2019.

[10] L. Yang, J. Cao, G. Liang, and X. Han, "Cost aware service placement and load dispatching in mobile cloud systems," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1440–1452, 2016.

[11] 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*, pp. 1–18, 2018.

[12] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: A network perspective," in *Conf. on Computer Communications (INFOCOM)*. IEEE, 2018, pp. 783–791.

[13] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 69–80.

[14] H.-J. Hong, P.-H. Tsai, A.-C. Cheng, M. Y. S. Uddin, N. Venkatasubramanian, and C.-H. Hsu, "Supporting internet-of-things analytics in a fog computing platform," in *Conf. on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 138–145.

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

[16] S. Challita, F. Paraiso, and P. Merle, "A Study of Virtual Machine Placement Optimization in Data Centers," in *Conf. on Cloud Computing and Services Science (CLOSER)*, 2017, pp. 343–350.

[17] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual Network Embedding: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[18] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," RFC 1654, 2015.

[19] B. Németh, B. Sonkoly, M. Rost, and S. Schmid, "Efficient service graph embedding: A practical approach," in *Conf. on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2016, pp. 19–25.