Optimized Application Deployment in the Fog[*]

Zoltán Ádám Mann[1], Andreas Metzger[1], Johannes Prade[2], and Robert Seidl[3]

[1] University of Duisburg-Essen, Essen, Germany
[2] Nokia, Munich, Germany
[3] Nokia Bell Labs, Munich, Germany

Abstract. Fog computing uses geographically distributed fog nodes that can supply nearby end devices with low-latency access to cloud-like compute resources. If the load of a fog node exceeds its capacity, some non-latency-critical application components may be offloaded to the cloud. Using commercial cloud offerings for such offloading incurs financial costs. Optimally deciding which application components to keep in the fog node and which ones to offload to the cloud is a difficult combinatorial problem. We introduce an optimization algorithm that (i) guarantees that the deployment always satisfies capacity constraints, (ii) achieves near-optimal cloud usage costs, and (iii) is fast enough to be run online. Experimental results show that our algorithm can optimize the deployment of hundreds of components in a fraction of a second on a commodity computer, while leading to only slightly higher costs than the optimum.

1 Introduction

Fog computing provides a decentralized infrastructure for supporting applications in domains like Internet of Things (IoT), cyber-physical systems, or smart manufacturing (Industry 4.0) [9, 13]. Such applications process data from distributed end devices. Processing these data in the end devices is often not feasible because of the very limited capacity (in terms of CPU, storage, battery) of the devices. Fog computing uses computational resources called *fog nodes* at the network edge, which offer higher capacity than end devices. A fog node thus can host applications that process data from end devices that are in the proximity of the fog node [1, 23]. This facilitates data processing with low latency, since modern communication technologies (like 4G and 5G) make the transfer of data from the end devices to nearby fog nodes very fast (e.g., 5G even offers real-time guarantees). This does not mean that all of the application's components need to reside in the fog node. Some application components, e.g., ones that are not latency-critical, may even be offloaded to the cloud instead of fog nodes, to benefit from the virtually unlimited computational capacity of the cloud [3].

This paper focuses on optimizing application component deployment between a fog node and the cloud. The fog node, which can be considered a small data

[*] Published in: *Proceedings of the 17th International Conference on Service-Oriented Computing* (ICSOC), pp. 283-298, 2019

center at the network edge, hosts a set of applications [10]. Each application consists of a set of components (e.g., microservices). The fog node offers virtualized, cloud-like resources for hosting the components, e.g., in virtual machines or containers. Although the computational capacity of a fog node is typically larger than that of end devices, it is still limited and much lower than that of the cloud [12]. If the load of the fog node exceeds its capacity, some application components thus may have to be offloaded to the cloud. However, this offloading entails two concerns. On the one hand, using a commercial cloud is associated with financial costs, both for the usage of cloud compute resources and for data transfers into and out of the cloud. On the other hand, for some of the components there may be an *affinity requirement* prescribing the component to remain in the fog node, e.g., because the component is critical in terms of latency or because the component deals with sensitive data that must not be uploaded to the cloud due to data protection reasons [18]. Deciding which components to move to the cloud leads to a complex optimization problem, in which capacity and affinity constraints have to be satisfied while costs stemming from using the cloud are minimized. In addition, optimizing application deployment is not a one-off activity. The deployment should be re-optimized regularly during operation, e.g., when a new application is added or an application is removed, the load on an application changes, cloud prices change, etc. After such events, the deployment of all applications may be re-optimized. Components can be migrated between the fog node and the cloud to adapt the deployment in these cases [24]. This requires the optimization algorithm to be fast enough to be used online.

This paper makes the following contributions:
(i) We formalize the problem of optimizing the deployment of application components between a fog node and the cloud.
(ii) We devise a heuristic algorithm (FogPart) for the problem. The result of the algorithm always satisfies the capacity and affinity requirements, whenever they can be satisfied. FogPart is a significantly extended version of the Kernighan-Lin algorithm for balanced graph partitioning [11]. FogPart is a fast heuristic that iteratively improves an existing partition by a sequence of local changes, while also being able to escape local optima.
(iii) We demonstrate the applicability of our algorithm by applying it to the smart manufacturing case study "Factory in a Box".
(iv) We experimentally evaluate the effectiveness of FogPart in terms of the resulting cloud usage cost and the algorithm's execution time and compare it with an exact algorithm and another heuristic.

The results show that the cost of the deployment found by FogPart is on average only 2.1% higher than the results of the exact algorithm. At the same time, FogPart is orders of magnitude faster, taking less than 300ms on a commodity computer to optimize the deployment of 450 components. FogPart delivers near-optimal results very quickly, making it applicable to practical use.

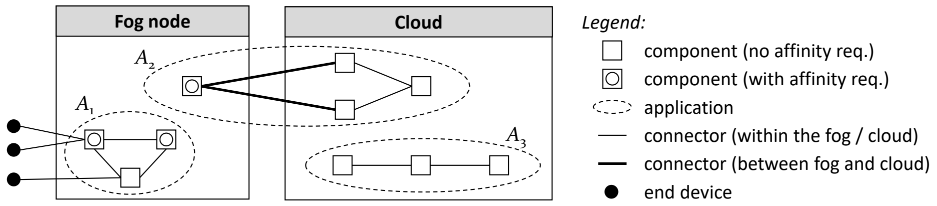

Fog node
Cloud
A_2
A_1
A_3
Legend:
component (no affinity req.)
component (with affinity req.)
application
connector (within the fog / cloud)
connector (between fog and cloud)
end device

Fig. 1: Schematic example of applications deployed in a fog node and the cloud

2 Problem formalization

Fig. 1 gives a schematic overview of the addressed problem. We are given a set $\mathcal{A}$ of applications to be deployed. An application $A \in \mathcal{A}$ is represented by an undirected graph $A = (V_A, E_A)$, where V_A is the set of components of application A and E_A is the set of connectors among the components. V_D denotes the set of end devices connected to the fog node, and E_D is the set of connectors between end devices and application components. The set of all end devices and components is $V = V_\mathrm{D} \cup \bigcup\{V_A : A \in \mathcal{A}\}$. The set of all connectors between end devices and components as well as among components is $E = E_\mathrm{D} \cup \bigcup\{E_A : A \in \mathcal{A}\}$.

For a component $v \in V$, $p(v) \in \mathbb{R}^+$ is the compute capacity required by v. As an example, this can be the number of CPU cores required by v. The predicate $s(v)$ is used to model the affinity requirement. The predicate is true if and only if v must remain in the fog node. If $v \in V_\mathrm{D}$, then $p(v) = 0$ and $s(v) = \text{true}$. For a connector $e \in E$, $h(e) \in \mathbb{R}^+$ denotes the amount of data exchanged along e.

$P \in \mathbb{R}^+$ denotes the compute capacity (e.g., number of CPU cores) of the fog node. The cost of renting a compute unit (vCPU) in the cloud is denoted by c_1, the unit cost of data transfer between the fog node and the cloud by c_2.

A *deployment* is a function $d : V \to \{\text{fog}, \text{cloud}\}$ that maps each component to either the fog node or the cloud. We use $\varrho(d)$ to denote the total compute capacity occupied in the fog node by deployment d: $\varrho(d) = \sum_{v \in V,\ d(v)=\text{fog}} p(v)$. A *valid* deployment must respect the following constraints:

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

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

Constraint (1) ensures that the total compute power required by the components in the fog node does not exceed the capacity of the fog node. Constraint (2) ensures that all affinity requirements are observed.

Our aim is to find a solution that minimizes the financial cost. For a deployment d, the set of connectors between the fog node and the cloud is defined as $E(d) = \{uv \in E : d(u) \neq d(v)\}$. Then, the cost of deployment d is:

$$\text{cost}(d) = \sum_{v \in V,\ d(v)=\text{cloud}} c_1 \cdot p(v) + \sum_{e \in E(d)} c_2 \cdot h(e), \tag{3}$$

where the first term is the cost of leased cloud resources and the second term is the cost of data transfers between the fog node and the cloud. Hence our aim is to minimize (3) while satisfying (1) and (2).

For a connector between an end device v_1 and a component v_2, $s(v_1) =$ true ensures that v_1 cannot be moved to the cloud. If $d(v_2) =$ cloud, then the connector crosses the boundary between fog and cloud, and thus it contributes to the costs in the second term of (3), otherwise it does not.

The deployment must be adapted in three cases: (i) when a new application is added, (ii) when an application is removed, (iii) when something changes in the deployed applications or in their environment.

Note that, in contrast to approaches aiming at placing one application on several fog nodes [4], our problem formulation focuses on the placement of a set of applications in a single fog node and the cloud. This is why our problem formulation differs from others in the literature (e.g., we use affinity requirements to ensure that latency-critical components are placed in the fog node, instead of working with application deadlines). Our approach fits the needs of a provider of an edge data center (an example application scenario is presented in Sec. 4).

3 The FogPart algorithm

The FogPart algorithm we propose to address the above optimization problem works on a model of the system. This means that the algorithm tentatively allocates and moves the components between the fog node and the cloud. Only after the algorithm terminates, the best found configuration is enacted by actually carrying out the necessary allocations and migrations.

When a new application is added, FogPart first places each new component v as follows: if $s(v) =$ true (i.e., v is subject to an affinity requirement), then v is placed in the fog node, otherwise in the cloud. Afterwards, the algorithm re-optimizes the deployment. When an application is removed, all its components are removed from the deployment, and a re-optimization is carried out. When there is a change in the deployed applications (e.g., in the CPU requirements of some components, the amount of data transfer between components, the affinity requirements of components) or in the environment (e.g., in the unit price of using the cloud or the capacity of the fog node), FogPart first ensures that the affinity requirement continues to hold by moving any affected component from the cloud to the fog node, and then performs re-optimization.

Re-optimization works the same way in each case. Re-optimization is based on iterative improvement: it starts from a – not necessarily valid – deployment and tries to improve it (making it valid and decreasing its cost) through a series of local changes. In each step, one component is moved either from the fog node to the cloud or vice versa; however, if the current deployment violates the capacity constraint, then only moves from the fog node to the cloud are allowed. The idea of the algorithm is to move the component leading to the highest decrease in cost, captured by the *gain* of the components.

Definition 1. *Let d be a deployment and $v \in V$ a component. Let d' be the deployment obtained from d by moving v. Then, given deployment d, the gain of moving v is defined as*

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

The algorithm prefers moves with higher gain values. To escape local optima, the move with highest gain is made even if its gain is negative, i.e., the move increases the cost (except if the gain is $-\infty$). To avoid infinite loops, each component may be moved only once during a re-optimization. When no further move is possible, the deployment with the lowest cost that was encountered during the algorithm is taken as the resulting new deployment.

The above re-optimization procedure in FogPart is an extended version of the Kernighan-Lin (KL) algorithm for balanced graph partitioning [11, 17]. Variants of the KL algorithm have been successfully applied to different partitioning problems [16, 19–21], thanks to the fact that it is a fast heuristic which can escape local optima. Applying the KL algorithm to our optimization problem required several extensions, since the original algorithm supports only edge costs, whereas our problem also contains costs related to vertices, as well as hard constraints on capacity and affinity, which are not supported by the original algorithm.

Algorithm 1 Deployment re-optimization

```
1: procedure RE-OPTIMIZE($d$)
2:     bestDeployment $\leftarrow d$
3:     bestCost $\leftarrow cost(d)$
4:     $L \leftarrow \{v \in V : \neg s(v)\}$
5:     end $\leftarrow (L = \emptyset)$
6:     while $\neg$end do
7:         bestGain $\leftarrow -\infty$
8:         for $v \in L$ do
9:             if $\varrho(d) \leq P$ or $d(v) =$ fog then
10:                $g \leftarrow$ GAIN(d,v)
11:                if $g >$ bestGain then
12:                    bestComp $\leftarrow v$
13:                    bestGain $\leftarrow g$
14:                end if
15:            end if
16:        end for
17:        if bestGain $> -\infty$ then
18:            forced $\leftarrow (\varrho(d) > P)$
19:            flip $d$(bestComp)
20:            $L$.remove(bestComp)
21:            if forced or $cost(d) <$ bestCost then
22:                bestDeployment $\leftarrow d$
23:                bestCost $\leftarrow cost(d)$
24:            end if
25:        end if
26:        end $\leftarrow (L = \emptyset$ or bestGain $= -\infty)$
27:    end while
28:    $d \leftarrow$ bestDeployment
29: end procedure
```

Algorithm 2 Calculation of the gain of moving a component

```
1: procedure GAIN($d, v$)
2:     if $d(v) =$ fog then
3:         $r \leftarrow -c_1 \cdot p(v)$
4:     else if $\varrho(d) \leq P$ and $\varrho(d) + p(v) > P$ then
5:         return $-\infty$
6:     else
7:         $r \leftarrow c_1 \cdot p(v)$
8:     end if
9:     for $vw \in E$ do
10:        if $d(v) = d(w)$ then
11:            $r \leftarrow r - c_2 \cdot h(vw)$
12:        else
13:            $r \leftarrow r + c_2 \cdot h(vw)$
14:        end if
15:    end for
16:    return $r$
17: end procedure
```

A more detailed description of the re-optimization procedure is given in Algorithm 1. The algorithm starts by setting "bestDeployment" and "bestCost" to the current deployment respectively its cost (lines 2-3). The list L contains the components that may be moved. In line 4, L is initialized to the set of all components without affinity requirements; the components with affinity requirements are not movable since they must remain in the fog node. In each iteration, one component is moved and it is removed from L (line 20); the procedure ends if L becomes empty, as captured by the Boolean variable "end" (lines 5, 6, 26).

In each iteration, first the component to be moved is determined ("bestComp"). For that purpose, "bestGain" is initialized to $-\infty$ (line 8), and then all movable components are checked (lines 8-16). Line 9 ensures that moving a component from the cloud to the fog node is not considered if the fog node is already overloaded. Lines 10-14 determine the component with the highest gain. If an allowed move is found, then it is performed (line 19) and the corresponding

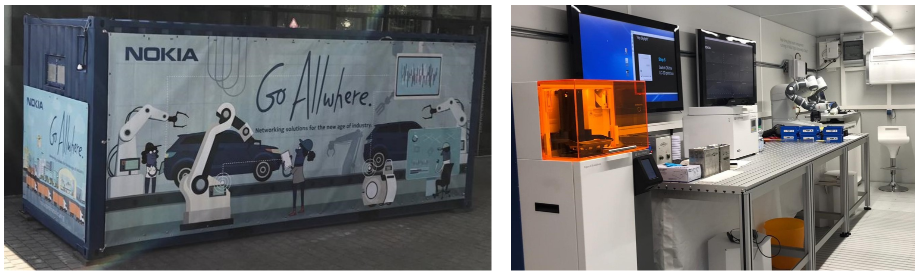


Fig. 2: Factory in a Box (FiaB): outside and inside view

component is removed from L (line 20). If the fog node was overloaded before the move, then the move is forced to be from the fog node to the cloud, as captured by the Boolean variable "forced". In this case, "bestDeployment" and "bestCost" are certainly updated with the changed deployment and its cost, otherwise they are updated only if the changed deployment is better than the best deployment encountered so far in terms of costs (lines 18, 21-24). The loop ends if there are no more movable components ($L = \emptyset$) or there are no valid moves, i.e., there are only moves that would invalidate the deployment ("bestGain" $= -\infty$) (line 26). Finally, the best deployment found is chosen (line 28).

The gain of a component is computed by Algorithm 2, in line with Definition 1. If the component v is currently in the fog node, then moving it to the cloud would increase costs by $c_1 \cdot p(v)$ (lines 2-3). If v is in the cloud, then moving it to the fog node would decrease costs by the same amount (lines 6-7). However, if the move violates the capacity constraint of the fog node, then the move is not allowed, resulting in a gain of $-\infty$ (lines 4-5). In lines 9-15, the connectors of v are investigated. For a connector vw, if v and w are either both in the cloud or both in the fog node, then the move would increase costs by $c_2 \cdot h(vw)$ (lines 10-11); otherwise, it would decrease costs by the same amount (lines 12-13).

4 Case study

To demonstrate the applicability of our approach and illustrate its operation, we applied it to a case study from the smart manufacturing domain, called "Factory in a Box" (FiaB). FiaB is an innovative factory solution, representing a complete production environment, integrated in a standard 20-feet freight container (see Fig. 2). It can host many different types of production lines, ranging from chemical processes, electronic device manufacturing to consumer goods. It accommodates a heterogeneous internal communication infrastructure, including novel mobile and fixed telecommunication technologies (e.g., private LTE and 5G) to serve various applications in the Industrial IoT environment. In addition to the computing capabilities within the FiaB (which we consider as the fog node), it connects to a cloud infrastructure using a public network.

Table 1: Characteristics of components in the FiaB case study

Application	Component	Required CPU cores	Affinity Req.
A_1	AM task manager	1	no
	iWh manager	1	no
	Robot control	1	yes
	Manual assembly SW	2	no
	Order management	2	no
	Supply management	2	no
	Tool management	1	yes
	Process management	1	yes
	ERP system	2	no
A_2	FiaB remote management	1	no
	Shop floor management	1	yes
A_3	Sensor evaluation SW	1	no
	Sensor dashboard	1	no

Table 2: Characteristics of connectors in the FiaB case study

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

For managing the production, multiple applications are needed. The characteristics of the application components respectively the connectors are shown in Table 1[4] and Table 2. The unit costs of compute resources and of data transfers to and from the cloud are determined based on Amazon EC2 pricing[5]. The hourly rental fee of a t2.small instance is USD 0.023, leading to a daily fee of USD 0.552, which is used as c_1. The transfer of 1GB of data to or from Amazon EC2 costs USD 0.09, which is used as c_2. The fog node has 12 CPU cores.

Running FogPart to add application A_1, first the components with affinity requirements (Robot control, Tool management, Process management) are put into the fog node and all other components are tentatively put into the cloud. Then, Algorithm 1 is executed to optimize the deployment. Algorithm 1 moves

[4] Abbreviations: AM = Additive Manufacturing, iWh = inbound Warehouse, VR/AR = virtual reality / augmented reality, ERP = Enterprise Resource Planning

[5] https://aws.amazon.com/ec2/pricing/on-demand/

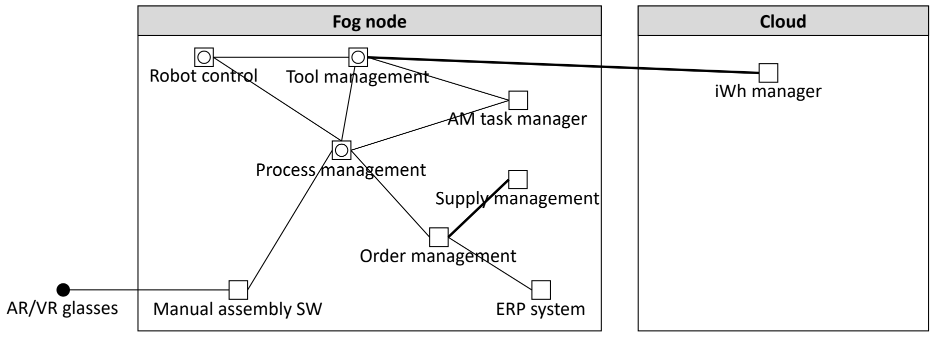


Fig. 3: Deployment of the first application in the FiaB case study

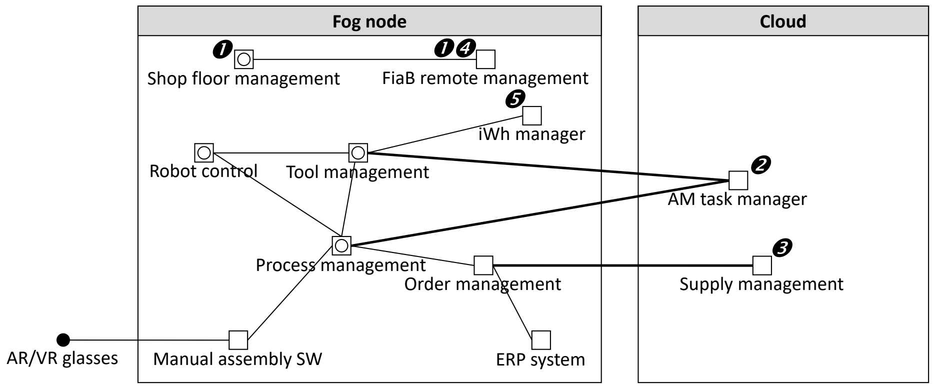


Fig. 4: Deployment of the second application. The numbers show the order in which the components are allocated and moved by FogPart

five further components from the cloud to the fog node, until the capacity of the fog node is exhausted, leading to the deployment shown in Fig. 3.

When application A_2 is deployed, first its component with an affinity requirement (Shop floor management) is put into the fog node and the other component (FiaB remote management) into the cloud. When re-optimizing the deployment, FogPart is confronted with an invalid deployment requiring 13 CPU cores in the fog node. Hence, FogPart first makes a forced move: the AM task manager is moved from the fog node to the cloud. This way, the deployment becomes valid, and it even reaches a local optimum: only moves from the fog node to the cloud are possible, which increase costs. FogPart makes one of these worsening moves: the Supply management is moved from the fog node to the cloud. As it turns out, this worsening move pays off: it frees up 2 CPU cores in the fog node, so that in the next two steps the FiaB remote management and iWh manager components can be moved to the fog node. The resulting deployment is better than the local optimum found earlier, as the heavy traffic between the Shop floor management and FiaB remote management components does not leave the fog

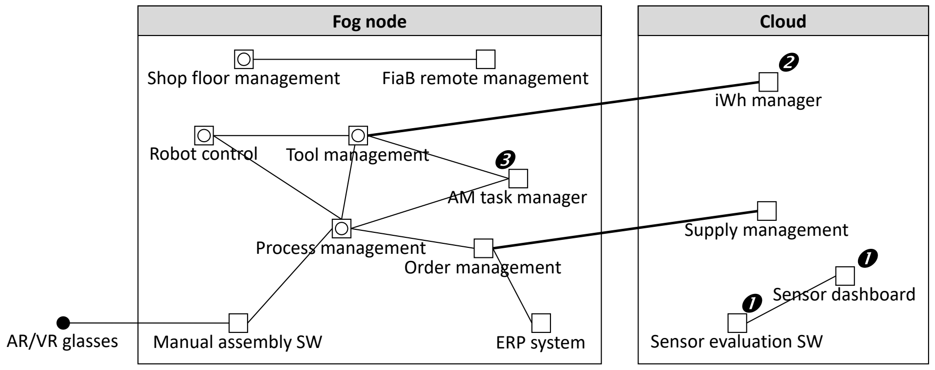


Fig. 5: Deployment of the third application (numbering as in Fig. 4)

node anymore. FogPart tries further moves but they do not lead to lower costs, hence the deployment shown in Fig. 4 is activated in the end.

When deploying application A_3, the new components (Sensor evaluation SW, Sensor dashboard) are put into the cloud. Since the capacity of the fog node is exhausted, only worsening moves are possible. The algorithm moves the iWh manager from the fog node to the cloud, after which it becomes possible to move the AM task manager from the cloud to the fog node. This leads to a better deployment, which also further moves cannot improve. In fact, the resulting deployment, which is shown in Fig. 5, is optimal.

These examples illustrate how FogPart continually ensures satisfaction of the requirements, and at the same time optimizes costs. In particular, the examples show how FogPart can escape local optima.

5 Experimental evaluation

To evaluate the costs of the solutions delivered by the FogPart algorithm as well as its execution time, we experimentally compare the performance of FogPart to two competing algorithms:
(i) Solving the optimization problem with an integer linear programming (ILP) solver as a typical example of an exact algorithm.
(ii) A simple heuristic based on the first-fit (FF) principle, as a typical example of a greedy algorithm. FF first deploys all components with affinity requirements in the fog node. The remaining components are deployed in the fog node if they fit, and otherwise in the cloud.

We implemented the three algorithms as a Java program that we made publicly available[6]. The experiments were performed on a Lenovo ThinkPad X1 laptop with Intel Core i5-4210U CPU @ 1.70GHz and 8GB RAM. The ILP-based algorithm uses the Gurobi Optimizer, version 7.0.2, as an external solver. The ILP solver was executed in single-threaded mode with a timeout of 60 seconds.

We simulated the following scenario:

[6] https://sourceforge.net/p/vm-alloc/hybrid-deployment/

– 10 applications are randomly generated with the following parameters:
  - $|V_A| = 30$
  - $p(v)$ is uniformly chosen from $\{1, 2, 3, 4\}$ for each $v \in V_A$
  - $s(v)$ is true with probability 0.1 for each $v \in V_A$
  - (V_A, E_A) is a complete graph
  - $h(e)$ is uniformly chosen from $[0.0, 3.0]$ for each $e \in E_A$

– Starting with $\mathcal{A} = \emptyset$, the applications are added one by one in the first 10 steps. Afterwards, 10 change steps are carried out, and finally the applications are removed one by one. Each change step performs one of the following actions (each with equal probability):
  - For each application, pick 3 random components and either increase or decrease their number of CPU cores by 1.
  - For each application, pick a random component v and let $s(v) := \neg s(v)$.
  - For each application, pick 10 random connectors and change their traffic intensity by multiplying with 2 or 0.5.
  - Change c_1, the unit cost of compute resources, by either increasing or decreasing it by 10%.

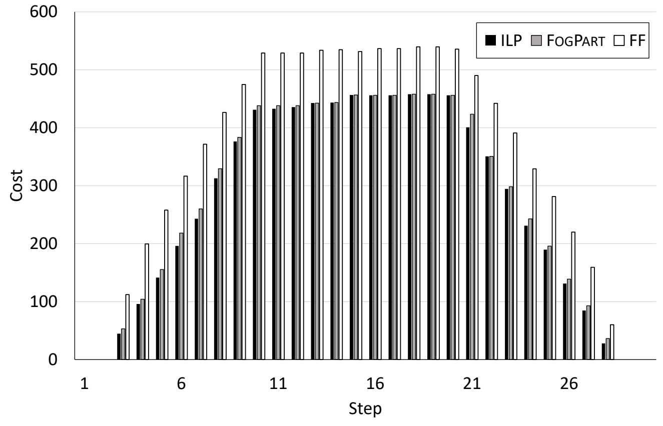
– As before, $c_1 = 0.552$ and $c_2 = 0.09$ (in line with Amazon EC2 pricing)

– $P = 150$

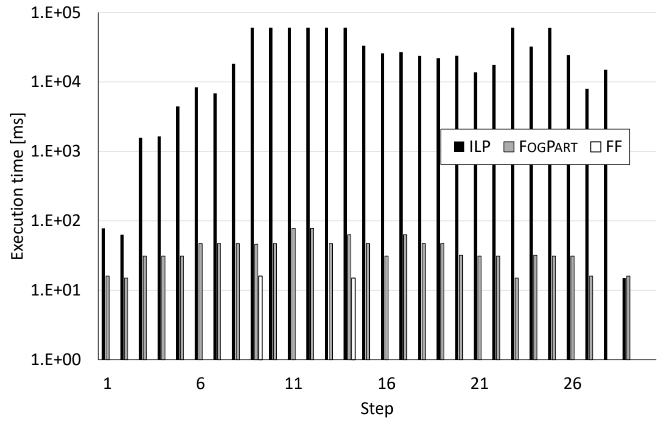
Fig. 6a shows the costs achieved by the three algorithms after each algorithm call. As expected, the costs monotonously increase in the first 10 steps and decrease in steps 20-30. In steps 1-2 and 29-30, it is possible to deploy all components in the fog node, leading to 0 costs; this optimal deployment is found by all algorithms. In the other steps, some components must be deployed in the cloud, leading to non-zero costs. Consistently across all steps 3-28, the results of FogPart are only slightly higher than those of the ILP-based algorithm, whereas the FF algorithm yields significantly higher costs. The costs achieved by FogPart are on average 2.19% higher than the costs achieved by the ILP-based algorithm; the costs of FF are 29.32% higher than those of ILP.

Fig. 6b shows the execution time of the three algorithms in each step. The time is shown in milliseconds, using logarithmic scale. The execution time of the ILP-based algorithm is consistently significantly higher than the execution time of the two heuristics. The average execution time of the ILP-based algorithm is roughly 26 seconds, while the average execution time is only about 36 milliseconds for FogPart and 1 millisecond for FF. In 8 cases, the execution time of the ILP-based algorithm reaches the timeout threshold of 60 seconds.

To evaluate scalability, we repeated the same call sequence as above, with varying number of components per application. In Fig. 7a, we report the total costs achieved by the three algorithms aggregated along the whole call sequence of adding, changing, and removing 10 applications. The number of components per application increased from 15 to 45 in increments of 5, thus leading to 150-450 components within a call sequence. Consistently across all application sizes, the cost of the deployments found by FogPart is only slightly higher than the costs achieved with ILP, whereas the costs achieved by FF are significantly higher. On average, FogPart leads to 2.1% higher costs than ILP, whereas FF



(a) Financial costs

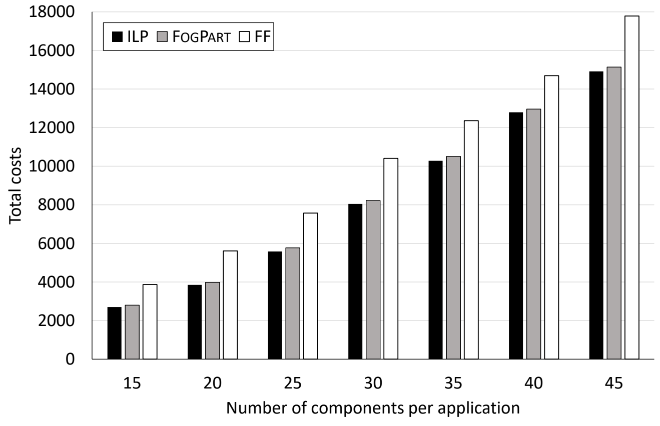


(b) Algorithm execution time (logarithmic scale)

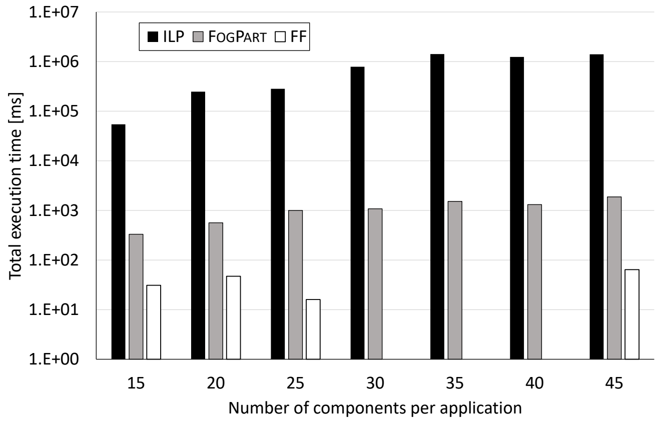
Fig. 6: Results of first adding, then changing, and finally removing 10 applications

leads to 24.3% higher costs than ILP. Interestingly, as the number of components grows, the relative difference between the algorithms' results decreases. This is because, as the number of components grows, also the number of components with affinity requirements grows, using up an increasing part of the fog node's capacity, and leaving less optimization opportunities for the deployment of the components without affinity requirements. E.g., when each application consists of 45 components, the expected number of CPU cores needed by the components with affinity requirements is 112.5, using up 75% of the capacity of the fog node.

Fig. 7b shows the total execution time – aggregated over the whole call sequence – of the three algorithms (note the logarithmic scale of the vertical axis). The execution time of FF is very low (tens of milliseconds for the whole call sequence), that of FogPart is somewhat higher but still quite low (less than 2 seconds in each case for the whole call sequence in total), and that of



(a) Financial costs



(b) Algorithm execution time (logarithmic scale)

Fig. 7: Impact of increasing the number of components

ILP is much higher (more than 20 minutes for a call sequence when the number of components is over 300). As the number of components grows, the execution time of both FogPart and ILP tends to grow. However, the growth rate is very different in the two cases. When the number of components grows from 150 to 450 – a threefold increase – the execution time of FogPart increases by a factor of 5.7, suggesting a moderate polynomial complexity. At the same time, the execution time of ILP grows by a factor of 26, suggesting an exponential execution time, damped down by the timeout of 60 seconds per run.

6 Related work

Some approaches have already been proposed in the literature for the optimized deployment of applications in the fog [4]. Mahmud et al. proposed a heuristic to allocate application components in a multi-layer fog system [14]. Taneja and Davy developed a greedy algorithm for placing application modules in the cloud and on fog nodes [27]. Skarlat et al. devised a genetic algorithm for optimizing the deployment of IoT applications on fog nodes [26]. Da Silva et al. considered the deployment of distributed stream processing applications on cloud and edge resources with the aim of minimizing application latency [25]. Cai et al. addressed the deployment of complex event processing applications on edge resources with the aim of minimizing the average application latency [5]. Mouradian et al. use tabu search to minimize the makespan of applications consisting of virtual network functions in the context of mobile fog nodes [22]. However, these approaches suffer from some serious limitations. First, some approaches support only applications with a special structure (cycles of four vertices [14], series-parallel graphs [25, 22], or directed acyclic graphs [5]). In contrast, our algorithm works for any application topology. Second, some approaches do not consider the costs of using the cloud at all [5, 25], or do not take data transfers between application components into account [26, 27], which, however, can lead to significant costs. In contrast, our algorithm explicitly minimizes the costs of using the cloud, including costs for both compute resources in the cloud and data transfer between the fog node and the cloud. Third, some approaches were based on simple greedy algorithms [14, 27] that consider only one application at a time and deploy its components sequentially. In contrast, our algorithm optimizes the deployment of all applications together, which increases the probability of finding overall good solutions, and uses special techniques to escape local optima.

Similar problems also arise when optimizing the deployment of applications in hybrid clouds. Several authors investigated the problem of scheduling a workflow using the resources of a hybrid cloud [2]. Chopra et al. proposed an algorithm for minimizing costs while respecting a given deadline [8]. Chang et al. proposed an agent-based mechanism to continually re-optimize the deployment of the jobs of a workflow in a hybrid cloud [7], while Zhu et al. addressed the scheduling of deadline-constrained workflows with stochastic tasks [29]. Another related area is the allocation of massively parallel tasks using the resources of a hybrid cloud. Van den Bossche et al. addressed the allocation of tasks to hybrid clouds taking into account application deadlines and cloud resource costs [28]. Candeia et al. aimed at maximizing profit, taking into account the benefit of finishing a set of compute tasks early and the costs of using cloud resources [6]. Malawski et al. used mixed integer nonlinear programming to allocate tasks to hybrid cloud resources, subject to deadline constraints, minimizing costs [15]. In all these papers, the communication structure between tasks is either constrained to be acyclic, which is an unrealistic assumption for many applications, or not considered at all. In contrast, our approach works with arbitrary communication topologies among the components of an application.

7 Conclusions

This paper addressed the problem of deploying application components on a fog node or in the cloud, such that components which need to be kept close to the end devices are deployed on the fog node, the capacity of the fog node is not overloaded, and the costs of using the cloud for computation and data transfer are minimized. We devised a heuristic for this problem. Our experimental results suggest that the results of our algorithm are close to optimal, while the algorithm is very fast so that it can be used online.

In the future, we aim to extend our approach to handle further constraints (e.g., modeling a more fine-grained control of latency) and optimization objectives (e.g., relating to energy consumption).

Acknowledgments. Research leading to these results received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements no. 731678 (RestAssured) and 731932 (TransformingTransport).

References

1. Abbas, Z., Li, J., Yadav, N., Tariq, I.: Computational task offloading in mobile edge computing using learning automata. In: IEEE ICCC. pp. 57–61 (2018)
2. Alkhanak, E.N., Lee, S.P., Rezaei, R., Parizi, R.M.: Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. Journal of Systems and Software 113, 1–26 (2016)
3. Bermbach, D., Pallas, F., Pérez, D.G., Plebani, P., Anderson, M., Kat, R., Tai, S.: A research perspective on fog computing. In: ICSOC. pp. 198–210 (2017)
4. Brogi, A., Forti, S., Guerrero, C., Lera, I.: How to place your apps in the fog – state of the art and open challenges. arXiv preprint, arXiv:1901.05717 (2019)
5. Cai, X., Kuang, H., Hu, H., Song, W., Lü, J.: Response time aware operator placement for complex event processing in edge computing. In: ICSOC. pp. 264–278 (2018)
6. Candeia, D., Araújo, R., Lopes, R., Brasileiro, F.: Investigating business-driven cloudburst schedulers for e-science bag-of-tasks applications. In: CloudCom. pp. 343–350 (2010)
7. Chang, Y.S., Fan, C.T., Sheu, R.K., Jhu, S.R., Yuan, S.M.: An agent-based workflow scheduling mechanism with deadline constraint on hybrid cloud environment. International Journal of Communication Systems 31(1), e3401 (2018)
8. Chopra, N., Singh, S.: Deadline and cost based workflow scheduling in hybrid cloud. In: ICACCI. pp. 840–846 (2013)
9. Dastjerdi, A.V., Buyya, R.: Fog computing: Helping the Internet of Things realize its potential. Computer 49(8), 112–116 (2016)
10. Deng, S., Xiang, Z., Yin, J., Taheri, J., Zomaya, A.Y.: Composition-driven IoT service provisioning in distributed edges. IEEE Access 6, 54258–54269 (2018)
11. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal 49(2), 291–307 (1970)
12. Lai, P., He, Q., Abdelrazek, M., Chen, F., Hosking, J., Grundy, J., Yang, Y.: Optimal edge user allocation in edge computing with variable sized vector bin packing. In: ICSOC. pp. 230–245 (2018)

13. Mahmud, R., Kotagiri, R., Buyya, R.: Fog computing: A taxonomy, survey and future directions. In: Internet of Everything, pp. 103–130. Springer (2018)
14. Mahmud, R., Ramamohanarao, K., Buyya, R.: Latency-aware application module management for fog computing environments. ACM ToIT 19(1), 9 (2018)
15. Malawski, M., Figiela, K., Nabrzyski, J.: Cost minimization for computational applications on hybrid cloud infrastructures. FGCS 29(7), 1786–1794 (2013)
16. Mann, Z.Á.: Partitioning algorithms for hardware/software co-design. Ph.D. thesis, Budapest University of Technology and Economics (2004)
17. Mann, Z.Á.: Optimization in computer engineering – Theory and applications. Scientific Research Publishing (2011)
18. Mann, Z.Á., Metzger, A.: Optimized cloud deployment of multi-tenant software considering data protection concerns. In: CCGRID. pp. 609–618 (2017)
19. Mann, Z.Á., Orbán, A., Farkas, V.: Evaluating the Kernighan-Lin heuristic for hardware/software partitioning. AMCS 17(2), 249–267 (2007)
20. Mann, Z.Á., Papp, P.A.: Formula partitioning revisited. In: 5th Pragmatics of SAT Workshop. vol. 27, pp. 41–56. EasyChair Proceedings in Computing (2014)
21. Mann, Z.Á., Papp, P.A.: Guiding SAT solving by formula partitioning. International Journal on Artificial Intelligence Tools 26(4), 1750011 (2017)
22. Mouradian, C., Kianpisheh, S., Abu-Lebdeh, M., Ebrahimnezhad, F., Jahromi, N.T., Glitho, R.H.: Application component placement in NFV-based hybrid cloud/fog systems with mobile fog nodes. IEEE JSAC 37(5), 1130–1143 (2019)
23. Nan, Y., Li, W., Bao, W., Delicato, F.C., Pires, P.F., Zomaya, A.Y.: A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems. Journal of Parallel and Distributed Computing 112, 53–66 (2018)
24. Ravindra, P., Khochare, A., Reddy, S.P., Sharma, S., Varshney, P., Simmhan, Y.: ECHO: An adaptive orchestration platform for hybrid dataflows across cloud and edge. In: ICSOC. pp. 395–410 (2017)
25. da Silva Veith, A., de Assuncao, M.D., Lefevre, L.: Latency-aware placement of data stream analytics on edge computing. In: ICSOC. pp. 215–229 (2018)
26. Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., Leitner, P.: Optimized IoT service placement in the fog. Service Oriented Comp. Appl. 11(4), 427–443 (2017)
27. Taneja, M., Davy, A.: Resource aware placement of IoT application modules in fog-cloud computing paradigm. In: IEEE IM. pp. 1222–1228 (2017)
28. Van den Bossche, R., Vanmechelen, K., Broeckhove, J.: Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In: IEEE CLOUD. pp. 228–235 (2010)
29. Zhu, J., Li, X., Ruiz, R., Xu, X.: Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources. IEEE TPDS 29(6), 1401–1415 (2018)