A Container-Driven Approach for Resource Provisioning in Edge-Fog Cloud

Hamid Mohammadi Fard[1](✉), Radu Prodan[2], and Felix Wolf[1]

[1] Department of Computer Science, Technical University of Darmstadt, Darmstadt, Germany
{fard,wolf}@cs.tu-darmstadt.de

[2] Alpen Adria-Universität Klagenfurt, Klagenfurt, Austria
radu@itec.aau.at

Abstract. With the emerging Internet of Things (IoT), distributed systems enter a new era. While pervasive and ubiquitous computing already became reality with the use of the cloud, IoT networks present new challenges because the ever growing number of IoT devices increases the latency of transferring data to central cloud data centers. Edge and fog computing represent practical solutions to counter the huge communication needs between IoT devices and the cloud. Considering the complexity and heterogeneity of edge and fog computing, however, resource provisioning remains the Achilles heel of efficiency for IoT applications. According to the importance of operating-system virtualization (so-called containerization), we propose an application-aware container scheduler that helps to orchestrate dynamic heterogeneous resources of edge and fog architectures. By considering available computational capacity, the proximity of computational resources to data producers and consumers, and the dynamic system status, our proposed scheduling mechanism selects the most adequate host to achieve the minimum response time for a given IoT service. We show how a hybrid use of containers and serverless microservices improves the performance of running IoT applications in fog-edge clouds and lowers usage fees. Moreover, our approach outperforms the scheduling mechanisms of Docker Swarm.

Keywords: Edge computing · Fog computing · Cloud computing · Resource provisioning · Containerization · Microservice · Orchestration · Scheduling

1 Introduction

The Internet of Things (IoT) has emerged by the rising number of connected smart technologies, which will remarkably affect the daily life of human beings in the near future. According to Cisco, 75 billion devices are expected to be connected to the Internet by 2025[1] in the future smart world. Nowadays, there

[1] https://www.cisco.com/c/en/us/solutions/internet-of-things/future-of-iot.html.

are countless IoT endpoints offloading their big data on the high performance resources of central clouds. In this traditional architecture, the raw data generated by IoT sensors are transferred to the cloud, which is in charge of filtering, processing, analyzing and persistently storing these data. After refining the data, the final results are transferred back to the IoT actuators to complete the cycle. The explosive amount of data produced by IoT sensors and the high computation demand for storing, transferring and analyzing these data are the new challenges of using centralized clouds that become a network and computational bottleneck.

A proposed solution to cover these challenges is a combination of edge, fog and cloud computing paradigms [4,11]. As shown in Fig. 1, the goal of this model, which we call it *edge-fog cloud* is to process and store data close to the producers and consumers instead of sending the entire traffic to the cloud resources. Therefore, the computation capacity for data analysis and application services will stay close to the end users, resulting in lower latency that is critically important for many types of real-time applications, such as augmented reality. As edge and fog computing are highly dynamic and increasingly complex distributed system paradigms with a high degree of heterogeneity [16], resource provisioning is one of the significant challenges in managing these architectures. Although edge and fog computing were suggested to deal with the response time and data latency of IoT applications, the edge and fog nodes are often not as strong as the cloud resources. Table 1 summarizes the main differences between edge, fog and cloud models by considering the features variations moving between the models.

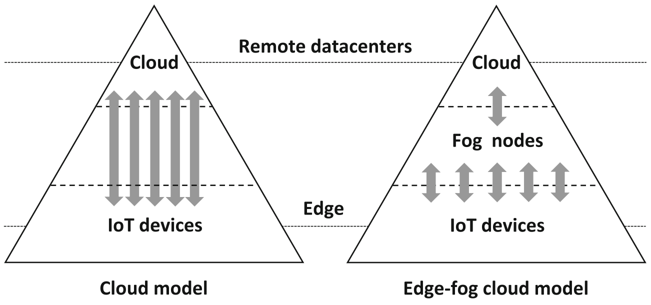


Fig. 1. Cloud vs. Edge-fog cloud: compared to the edge-fog cloud, execution of IoT applications in the cloud only causes much longer data transfer time.

In this paper, we discuss and analyze the efficiency of combining containers and serverless microservices compared to hardware virtualization for resource provisioning in an edge-fog cloud. We present that despite the limited resource capacity of edge and fog layers, the proximity of IoT nodes and computation resources in edge and fog computing reduces the communication cost of services and plays a remarkable role for achieving effective latency. By considering the available computation capacity, the proximity of these computation resources to

Table 1. Comparison of edge, fog and cloud models. Moving from cloud to IoT layer, node, network and data specications change considerably

	Specification	Change
Nodes	Number	Increasing
	Heterogeneity	Increasing
	Reliability	Decreasing
	Computation capability	Decreasing
	Mobility	Increasing
Network	Heterogeneity	Increasing
	Bandwidth	Decreasing
	Traffic	Increasing
Data	Volume	Increasing
	Persistence	Decreasing
	Distribution	Increasing

data producers and consumers and the dynamic nature of edge-fog cloud model, we propose a novel container orchestration mechanism to minimize the end-to-end latency of services. Our proposed mechanism selects the most adequate host to achieve the minimum response time for IoT services. Our scheduling mechanism can be implemented as a plugin module for any available orchestration framework such as Docker Swarm[2] and Kubernetes[3].

In Sect. 2, we first discuss how edge-fog cloud applications can benefit from containerization technology. Next in Sect. 3, we review the related work for resource provisioning problem in edge-fog cloud environment. We model the problem formally in Sect. 4. In Sect. 5, we propose our container orchestration approach, which is evaluated in Sect. 6. Finally, we conclude the paper in Sect. 7.

2 Containerization and Edge-Fog Cloud

Container as a service (CaaS) is relatively a new offering of almost all major cloud providers including Amazon Web Services, Microsoft Azure and Google Cloud Platform. Containerization is a lightweight kernel- or operating system (OS)-level virtualization technology [8]. A container is an isolated environment that includes its own process table structure, services, network adapters and mount points. As shown in Fig. 2, containers and virtual machines (VM) are two technologies for consolidation of hardware platforms. Containers are similar to VMs with a major difference that they run on a shared OS kernel. In contrast, traditional VMs (based on hardware-level virtualization) suffer from the overhead needed to separate the OS for individual VMs, which causes the waste of resources. Using the abstraction layer called containerization engine, libraries

[2] https://docs.docker.com/engine/swarm/.
[3] https://kubernetes.io/.

and application bundles no longer need complete and separated OS. Containers separate a single OS from isolated user spaces by using several OS features such as kernel namespaces and control group (cgroup). This type of isolation is more flexible and efficient than using hypervisor virtualized hardware (such as vCPU and vRAM) [5].

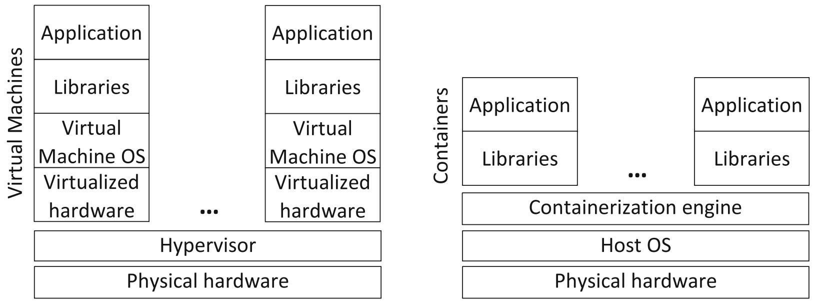


Fig. 2. Containers versus virtual machines architectural comparison.

In comparison to hardware-level virtualization, kernel-level virtualization benefits from several remarkable advantages:

- containers can be deployed faster;
- containers are more scalable than VMs;
- booting containers takes few seconds (or even milliseconds for cached images) instead of tens of seconds for VMs;
- containers are more portable across infrastructures;
- because of sharing kernel services, containers consume and waste less resources;
- images of containers are smaller and can be transferred or migrated faster;
- containers are usable in limited bandwidth environments;
- containerized services are cheaper than leasing VMs in public clouds.

Container technologies, such as LXC and LXD[4], have been around for more than a decade, but they got popularized by Docker[5], which proposed in 2013 a simple to use framework. Docker is an API around several open source Linux containers projects that wraps up a piece of software in a complete file system including code, runtime system tools and libraries.

Orchestration tools are responsible for placement and state management of containers in a cluster. Swarm is the native clustering solution in the Docker ecosystem and there are several third-party orchestration tools usable for Docker containers like Kubernetes and Apache Memos[6]. In fact, the main task of orchestrator is to choose a node out of all available cluster nodes for deploying a container, considering all its requirements (e.g. fast storage).

[4] https://www.linuxcontainers.org/.
[5] https://www.docker.com/.
[6] http://mesos.apache.org/.

Containerization is the major backbone technology used for microservice architectures. A *serverless microservice* is a package that consists of the entire environment required to run an application, including the application and its dependencies. This technology is not new either and has been promoted by the Heroku[7] as the initiator of this kind of services. A serverless microservice is a fine-grained service that can be run on demand on the cloud and has several features such as auto-scaling, pay-per-execution, short-lived, stateless functions and event-driven workflow. To gain these features, we need to transform monolithic applications into a microservices-oriented architecture, which is not always easy because of its complexity and the need of redesign in most cases.

The limited computation capability of edge and fog nodes forces the applications to be designed and decomposed as less resource-intensive and more lightweight services. Moreover, the mobility of the endpoints [3] in IoT networks (e.g. wearable devices, smart phones, car cameras) needs lightweight migration of services. Infrastructure agnosticism provided by containerization covers the ultra heterogeneity of resources in complex environments like edge-fog cloud. Using containers, the infrastructure becomes an application platform rather than plain data center hardware. Considering the advantages of containers and microservices, particularly their single-service, short-lived and lightweight nature and their ultra scalability, we motivated to use their combination efficiently for resource provisioning in an edge-fog cloud environment.

3 Related Work

IoT devices are usually simple sensors and embedded systems with low battery, low computation capacity and low bandwidth level. To efficiently execute IoT applications in an edge-fog cloud, resource provisioning and scheduling of services are of highest importance [3,14].

Because of dynamic nature of the IoT network, static scheduling and dynamic rescheduling of resources are not efficiently applicable in an edge-fog cloud that requires fully dynamic approaches [1]. Although the fog is assumed as a new distributed system extending the cloud, scheduling approaches such as [7] are inefficient and need to be customized to deal with the new challenges of an edge-fog cloud environment.

In [17], the authors proposed a time-shared provisioning approach for services. The main simplification in their model is neglecting the dynamic nature of fog environments.

In [15], the authors assumed that the edge and fog devices are powerful enough for hardware virtualization which is not always true. We discuss the inefficiency of hardware virtualization in our results in Sect. 6.

Fog computing extends the cloud boundaries such that the fog nodes can play the providers' role. An example implementation of a new cloud is the iExec project[8]. FogSpot [18] is a spot pricing approach for service provisioning of IoT

[7] https://www.heroku.com/.

[8] https://iex.ec/.

applications in fog computing, but ignores many challenges in such a market. For these new commercial computation models, we need to deal with many issues like reliability of resources and selfishness of the providers. Using game theory, we proposed a truthful market model [6] for execution of scientific applications in a cloud federation that can be extended easily for edge-fog cloud market.

Different models have been proposed for edge and fog computing [19,21] and all have the same crucial constraints for resource provisioning. Some works such as [17] miss implementation details. The authors of [20] proposed a container-based task scheduling model considering assembly lines in smart manufacturing.

The idea of using containerization is a controversial subject too. Some researches use containerization as a proper and efficient approach [12,13], while other works [2] claim that containers are inefficient in fog computing. In this paper we propose an edge-fog cloud model and a container orchestration approach, efficiently usable in such an environment.

4 System Model

In this section, we formally define our model including platform, application and problem models.

4.1 Platform Model

Usually two approaches are followed for implementing edge and fog computing models. In the first approach called cloud-edge, the public cloud providers with the help of internet and telecommunication service providers, extend their data centers in multiple point-of-presence (PoP) locations. Although this approach is widely used, it is costly and limited to special locations and services. In the second approach called device-edge, different organizations emulate the cloud services by running a custom software stack on their existing geo-distributed hardware. Any device including computation power and storage with connected network could be a fog or edge node. This approach is more affordable in many scenarios and can efficiently utilize the organizational in-house infrastructure. Inspired by these models, we propose our general Edge-Fog Cloud model, which is a hybrid combination of both cloud-edge and device-edge models and involves all other new computation models in this domain, such as dew and mist computing.

We assume a set of m geographically distributed non-mobile IoT devices (on the edge) denoted as $D = \{d_1, \ldots, d_m\}$, belonging to an organization. The fog layer contains a range of devices including network equipment (e.g. Cisco IOx routers), geo-distributed personal computers, cloudlets and micro- and mini-data centers. We model the fog layer by the set of n geo-distributed nodes, denoted as $F = \{f_1, \ldots, f_n\}$. The set of p leased virtual machines in different availability zones provided by the cloud federation providers is denoted by $C = \{c_1, \ldots, c_p\}$. In this model, we assume that $m \gg n \gg p$.

We assume that F is the in-house IT infrastructure of the organization (e.g. routers and local distributed data centers) and C is the public cloud infrastructure that needs to be leased by the organization on demand. An abstraction of our model is shown in Fig. 3. Based on this model, using the local IT infrastructure has no extra cost for the organization, but it needs to pay for using public infrastructure, which includes the cost of data transfer to and from the cloud and the cost of using cloud computation capacities.

The cloud resources in C can be used in two different ways; reserved in advance (by leasing virtual machine instances for example) or by calling serverless microservices. Therefore, to implement our edge-fog cloud model (cluster), we follow two approaches:

(a) model_a clusters all nodes available in $V = D \cup F \cup C$.
(b) model_b clusters only the nodes in $D \cup F$ (in other words $V - C$) and the cloud resources are used as serverless microservice calls.

In model_a, we need to regularly lease the cloud resources based on the cloud pay-as-you-go model (e.g. hourly-based virtual machines) and then to add the leased nodes to the cluster. In model_b we do not lease and reserve any resource in advance, but pay based on the number of service calls. We evaluate both implementations in Sect. 6 and compare their differences.

The network topology, connecting three layers (see Fig. 3), is modeled as a weighted directed graph $G(V, E)$ such that the set of all available nodes is the vector set V and the network links available between the nodes denotes the edge set E.

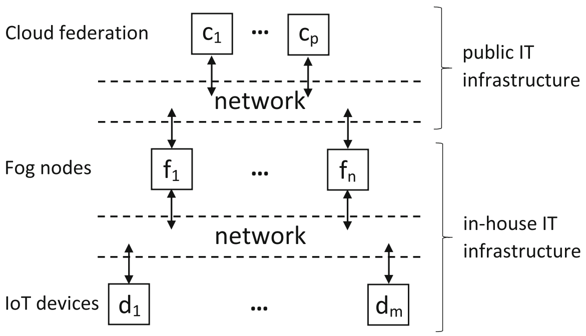


Fig. 3. Edge-fog cloud model: IoT, edge and fog nodes are in-house assets and the cloud data centers belong to an enterprise public cloud federation.

4.2 Application Model

Each IoT device $d_i \in D$ calls a set of services, denoted by $s_i = \{s_i^1, ..., s_i^{l_i}\}$ such that l_i is the number of services that might be called by d_i. The services are modeled as stateless containerized services on the edge. The set of all services required by D is $S = \{s_1, ..., s_m\}$. To sake of precise modeling, we need to notice that while there can be services shared by two devices d_i and d_j such that $s_i \cap s_j \neq \emptyset$, sharing stateless services has no impact on the model.

Theoretically, each service of S can run on cloud, fog and even IoT devices (edge), which have enough hardware to run the containerized services. Therefore each node $v_i \in V$ can be potentially a container host in this model.

Each service $s_i^j \in s_i$ is initiated by the IoT device d_i. To run the service s_i^j on a node $v_k \in V$, we need to transfer the required input data from d_i to v_k that will last $timeIn(s_i^j, v_k)$. After the execution, we need to transfer the output data from v_k to d_i that takes $timeOut(s_i^j, v_k)$. Consequently, the entire data transfer time between the device and the service's host is $timeInOut(s_i^j, v_k) = timeIn(s_i^j, v_k) + timeOut(s_i^j, v_k)$.

The service processing time for s_i^j on v_k lasts $timeProcess(s_i^j, v_k)$. If the service s_i^j cannot physically run on v_k, for instance because of need to special hardware or privacy issues, we define $timeProcess(s_i^j, v_k) = \infty$.

Because the services running on the cluster are containerized, we need to model the image transfer time of each container from a locally-implemented image repository by $timeImage(s_i^j, v_k)$. Depending on the service used frequency and the amount of available storage per host, the image may be cached on the host and thus, $timeImage(s_i^j, v_k) = 0$. To keep the model simple, however, we ignore this situation without loss of generality.

4.3 Problem Model

To place a service s_i^j called by the IoT device d_i on a cluster node v_k, we define the orchestration as the function $orchestration(s_i^j, v_k) : S \mapsto V$. Since the services are dynamically initiated by the IoT devices, we cannot assume a predefined and static communication network graph between S and V. In other words, the network graph for assigning each service to the cluster is a subgraph of G. A single and static network graph allows us to benefit from techniques such as *betweenness centrality* from graph theory to find the best placement of services [10]. However, considering the dynamic complexity and variety of the network graph, we need to apply other dynamic heuristic- or greedy-based approaches.

5 Minimizing End-to-End Latency Algorithm

In this section, we propose a novel resource provisioning approach for our proposed edge-fog cloud architecture based on a dynamic application-aware container orchestration called *Minimizing End-to-End Latency (METEL)*. The main goal of METEL is to run the services S on the cluster nodes V to minimize the round-trip time of each single service. Since scheduling of containers in serverless microservices provided by commercial public clouds (e.g. AWS Lambda) are controlled by the providers, we concentrate in METEL only on the user-level orchestration.

Transferring multi-hop distance for each chunk of data is timely and costly inefficient. On the other hand, clearly one cannot always expect that processing data on the adjacent nodes reduces the service delivery latency. Of course, the proximity of IoT nodes and computation resources reduces the network traffic and communication cost, however, a major challenge in the edge-fog cloud provisioning is the lack of powerful resources compared to the cloud data centers. To minimize the end-to-end latency, one should not simply rely on the proximity of nodes and minimize the data movement only. For container orchestration, in addition to the proximity, we need to define the effective latency by considering the processing time and the image transfer delay of each service. Since the edge and fog nodes are not rich capacity resources, there is always a tradeoff between the available capacity and the proximity of data producers and consumers to the computational resources.

Our scheduling mechanism needs least modification in the available orchestration frameworks such as Docker Swarm and Kubernetes and can be implemented as a plugin besides any orchestration module. For instance, as displayed in Fig. 4, using Docker APIs, METEL extracts the cluster information from the discovery service in Swarm and, after making the decision about the most adequate worker node, can justify the constraints in Swarm (e.g. by using affinity filters, placement constraints or host labels) such that the container is hosted on the selected node.

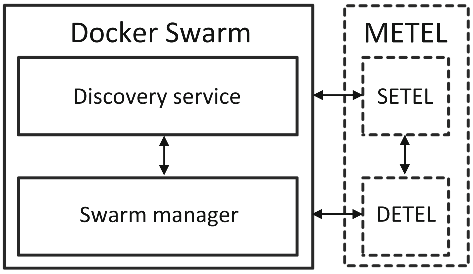


Fig. 4. METEL implementation in the Docker Swarm mode.

METEL includes two main modules: Algorithm 1, called SETEL, to calculate the *static end-to-end-latency* and Algorithm 2, called DETEL, to select the most adequate worker nodes for running the services, based on *dynamic end-to-end-latency*. The role of these two modules in METEL algorithm and their relation are shown in Fig. 5.

We declare a global two-dimensional matrix $setelMatrix$ to store the static end-to-end latency of each service $s_i^j \in S$ to each $v_k \in V$. The static end-to-end latency is the latency of running a service ignoring the dynamic load and availability of the resources, calculated offline based on the static available information. Algorithm 1 calculates the static end-to-end latency of services on each cluster node. In lines 2–6, the Dijkstra's algorithm calculates the shortest cycle from and to each IoT endpoint $d_i \in D$ which must pass through $v_k \in V$. Because of different up- and down-links between the devices, the send and

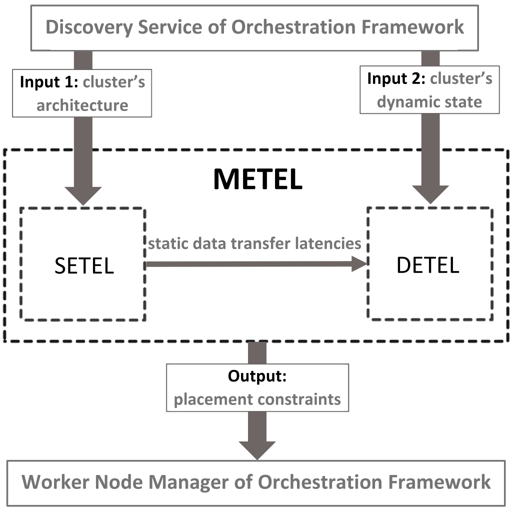

Discovery Service of Orchestration Framework
Input 1: cluster's architecture
Input 2: cluster's dynamic state
METEL
SETEL
static data transfer latencies
DETEL
Output: placement constraints
Worker Node Manager of Orchestration Framework

Fig. 5. METEL's inside. SETEL (Static End-to-End Latency) calculates the static latency. It runs once at launch time and is triggered and run by cluster changes. DETEL (Dynamic End-to-End Latency) adjusts the placement constraints to select the worker node for containers and runs in each orchestration decision.

receive communication paths between the IoT devices and fog nodes may not be the same. The three nested loops in lines 7–13 calculate the static-end-to-end latency of the services. The algorithm returns the calculated static end-to-end latency of all services on all nodes in $setelMatrix$. The matrix is updated whenever the resource discovery module in the container orchestration detects an infrastructure change, such as adding a new node or failing an already available node.

Dynamic end-to-end latency of each service is calculated by considering the static end-to-end latency and the dynamic status of the worker node. Algorithm 2 makes the final orchestration decision for each service s_i^j using $setelMatrix$ calculated by Algorithm 1, and returning the cluster node which provides the minimum effective latency for the service s_i^j. The function $delay(s_i^j, v_k)$ in line 4 returns the dynamic delay of the service s_i^j on the worker node v_k, obtained from runtime status of the nodes. Lines 3–10 find the host which provides the lowest completion time of the service s_i^j, and line 11 returns the selected host v_{min} as the final schedule decision.

The sequence diagram of launching a service is shown in Fig. 6. Upon requesting a service by an IoT device, the scheduler finds the proper host based on METEL, which justifies the builtin orchestrator constraints such that the service is hosted on the selected node. The rest of the container life-cycle is monitored and controlled by the orchestrator.

Algorithm 1: Static End-to-End Latency (SETEL).

```
Input: Set of services: $S$; Set of worker nodes: $V$
Output: Static end-to-end latency matrix: $setelMatrix$
1  begin
2      for $i \leftarrow 1$ to $m$ do
                                                  /* Iterate over IoT devices */
3          for $k \leftarrow 1$ to $|V|$ do
                                              /* Iterate over the cluster nodes */
4              $spf(d_i, v_k) \leftarrow Dijkstra(\textbf{from}: d_i, \textbf{to}: d_i, \textbf{must\_pass}: v_k)$   /* Find shortest
               cycle from $d_i$ through $v_k$ */
5          end
6      end
7      for $i \leftarrow 1$ to $m$ do
                                                  /* Iterate over IoT devices */
8          for $j \leftarrow 1$ to $l_i$ do
                                          /* Iterate over IoT device services */
9              for $k \leftarrow 1$ to $|V|$ do
                                                /* Iterate over cluster nodes */
10                 $setelMatrix(s_i^j, v_k) \leftarrow$
                   $timeProcess(s_i^j, v_k) + timeInOut(s_i^j, v_k) + Image(s_i^j, v_k)$
                   /* Calculate static latency of each service $s_i^j$ on each node $v_k$ */
11             end
12         end
13     end
14     return $setelMatrix$
15 end
```

Algorithm 2: Dynamic End-to-End Latency (DETEL).

```
Input: Service called by $d_i$: $s_i^j$; Set of worker nodes: $V$; Static end-to-end latency matrix:
       $setelMatrix$
Output: Worker node for running $s_i^j$: $v_{\min}$
1  begin
2      $detel_{\min} \leftarrow \infty$                  /* Dynamic latency of running service $s_i^j$ */
3      for $k \leftarrow 1$ to $|V|$ do
                                                /* Iterate over cluster nodes */
4          $detel \leftarrow setelMatrix(s_i^j, v_k) + delay(s_i^j, v_k)$     /* Calculate dynamic latency of
           running service $s_i^j$ on node $v_k$ */
5          if $detel < detel_{\min}$ then
                                  /* Find node with lowest dynamic latency */
6              $detel_{\min} \leftarrow detel$
7              $v_{\min} \leftarrow v_k$
8          else
9          end
10     end
11     return $v_{\min}$
12 end
```

5.1 Time Complexity Analysis

To analyze the time complexity of METEL, we need to analyze SETEL and DETEL algorithms separately.

As shown in Fig. 5, DETEL runs dynamically at each orchestration decision. The time complexity of Algorithm 2 is $\mathcal{O}(|V|)$, which is simply linear (as discussed in Sect. 4.1, $|V| = m + n + p$).

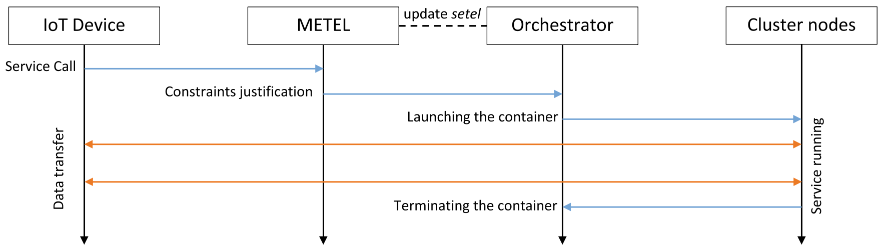


Fig. 6. METEL service orchestration timeline based on the sense-process-actuate model.

The complex part of METEL is SETEL module. As shown in Fig. 5, Algorithm 1 runs once at launch time and is triggered and run by cluster changes. Algorithm 1 includes two nested loops. Respecting the time complexity of Dijkstra's algorithm, which is $\mathcal{O}(|E| + |V| \cdot \log|V|)$, the time complexity of the first loop (2–6) is $\mathcal{O}(m \cdot |V| \cdot (|E| + |V| \cdot \log|V|))$. As discussed in Sect. 4.1, if we assume a fully connected network between all nodes in two adjacent layers of the graph G (between IoT and fog nodes and between fog and cloud nodes) then $|E| = n \cdot (m + p)$. The time complexity of the second loop (lines 7–13) is $\mathcal{O}(m \cdot \max\limits_{1 \le i \le m}(l_i) \cdot |V|)$, which will be dominated by the time complexity of the first loop.

About the time complexity of the first loop in SETEL, we need to notice several important facts in the real world problems:

- $|E| \ll n \cdot (m + p)$ because the nodes in two adjacent layers are not fully connected;
- as discussed in Sect. 4.2, for all nodes with $timeProcess(s_i^j, v_k) = \infty$, we do not need to run Dijkstra's algorithm.

Consequently, the final time complexity of Algorithm 1 is much lower than $\mathcal{O}(m \cdot |V| \cdot (|E| + |V| \cdot \log|V|))$. Moreover, we need to notice that the calculated time complexity for SETEL is for the first run of the algorithm at lunch time. Algorithm 1 will be also triggered and run by cluster changes but in this case, the update of $setelMatrix$ is only calculated for the cluster changes not for the whole cluster. In other words, the time complexity of Algorithm 1 to update $setelMatrix$ is much lower than the time complexity of first run of the algorithm. Therefore, in practice we could observe that METEL is really well scalable, even for enterprise organizations with large number of IoT devices and services.

6 Evaluation

Considering the variety and the number of resources in edge, fog and cloud layers, resource management in the edge-fog cloud is a complex task. The real-time need of many IoT applications makes this problem even more complicated.

Running comprehensive empirical analysis for the resource management algorithms in such a problem would be very costly, therefore, we rely on simulation environment.

For evaluating our approach, we ran an extensive set of experiments, based on the iFogSim [9] simulator for sense-process-actuate modeling of IoT applications. However, we needed to extend iFogSim to overcome some of its limitations required by our experiments. First, iFogSim implements a tree network structure (a hierarchical topology with direct communication possible only between a parent-child pair). To create a flexible network topology, we needed to replicate IoT devices for each gateway by extending the `Tuple` class. Second, iFogSim does not support containerization. To cover this, we extended the `AppModule` class, which is the entity scheduled on the fog devices. The simulation setup used in our experiments is summarized in Table 2.

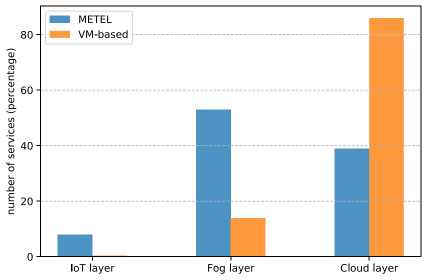
Table 2. Experimental simulation setup.

Entity	Entity specification
IoT devices	$m = 500$
	$10000 \leq computation\ capacity\ (MIPS^{a}) \leq 20000$
IoT services	$1 \leq \lvert s_i \rvert \leq 5$
	$3000 \leq service\ size\ (MI^{b}) \leq 30000$
	$20 \leq container\ image\ size\ (MB) \leq 100$
	$1 \leq data\ rate\ (MB/s) \leq 10$
Fog nodes	$n = 50$
	$15000 \leq computation\ capacity\ (MIPS) \leq 30000$
Cloud zones	$p = 5$
	$20000 \leq computation\ capacity\ (MIPS) \leq 80000$
Network	$1 \leq bandwidth(Mb/s) \leq 100$
	$1 \leq hops\ to\ fog \leq 5$
	$10 \leq hops\ to\ cloud \leq 15$

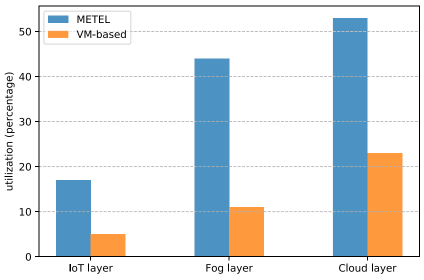
[a]MIPS: Million Instructions Per Second
[b]MI: Million Instructions

6.1 Containers Versus VMs

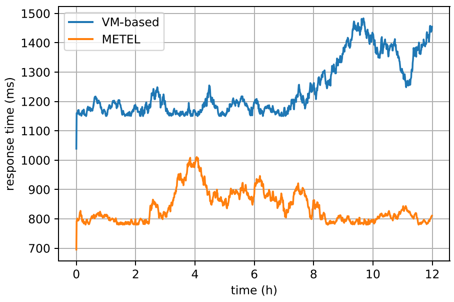
In the first part of the experiments (see Fig. 7), we analyzed the efficiency of containerization in edge-fog cloud resource provisioning and compared METEL with a VM-based provisioning approach using no containers. In the VM-based approach we launched separated VMs for isolated services, while for shared services we used a single shared VM. The VMs including services are launched at the start of the simulation and kept running across the entire evaluation time, which avoids the overhead of launching services. For launching VMs, we defined three priority levels: IoT devices, fog nodes and cloud. To launch a service, we first search in the IoT device layer. If there is no possibility to launch an IoT service, we search in the fog layer. Finally in case of not enough resources in the fog, we launch a VM in the cloud.



(a) Number of services running in different layers.



(b) Average utilization of different layers.



(c) Response times.

Fig. 7. Experimental comparison of METEL against a pure VM-based approach.

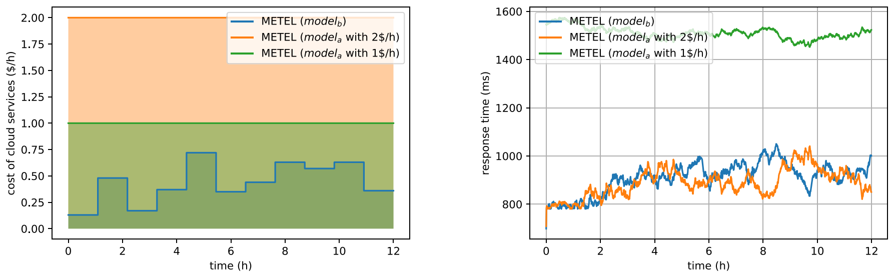
Figure 7a compares the number of services distributed across different edge-fog cloud layers. First, we observe that in a pure VM-based approach, the IoT resources are not rich enough to launch the VMs and no IoT device can provide a service. Because of the lightweight containers, METEL was able to run around 8% of the services on the IoT layer. Similarly, we observed that METEL executed around 57% of the services in the fog, in comparison to 10% by VM-based approach. In contrast, the pure VM-based approach run close to 90% of the services in the cloud layer, against around 40% run by METEL. Using no containerization, we not only spend a higher cost for leasing cloud resources, but also increase the latency. As we discussed before, the response time is important in comparing the final results, not the latency time.

Figure 7b represents the average utilization of nodes on the IoT, fog and cloud layers, which is considerably lower using a VM-based approach compared to METEL. As expected, the lightweight containers improve the consolidation of the resources and increase the average utilization in all three layers.

Figure 7c shows that METEL attains much better response time compared to the VM-based approach because of lower latency time and close proximity of producers and consumers.

6.2 Serverless Versus Containers

In the next experiments, we compare two implementations of METEL based on the two proposed models $\mathrm{model_a}$ and $\mathrm{model_b}$, discussed in Sect. 4, for a period of 12 h. The motivation for this analysis is to evaluate the efficiency of using serverless microservice implementation of services compared to calling remote containerized services on the leased cloud VMs. For this experiment, we simulated the service prices based on the AWS EC2[9] and AWS Lambda pricing[10] models.



(a) Cost of running cloud resources. (b) Response time.

Fig. 8. Experimental comparison of $\mathrm{model_a}$ and $\mathrm{model_b}$.

Figure 8a shows the monetary cost of our two implementations of METEL. For $\mathrm{model_a}$, we present the results of running the cloud services with \$1 and \$2 per hour (or \$12 and \$24 for 12 h). The services run as containerized services on these leased cloud resources. In the $\mathrm{model_b}$, we do not lease VMs on the cloud, but use serverless microservice calls instead. As shown in Fig. 8a, $\mathrm{model_b}$ has remarkably lower cloud expenses using serverless microservice calls. Furthermore, the total cost in $\mathrm{model_a}$ is even higher than the measured cost because we ignored the data transfer cost in our model. As shown in Fig. 8b, serverless microservices do not have a remarkable overhead in response time compared to calling services directly on leased VMs. In addition, not leasing enough resources in $\mathrm{model_b}$ dramatically increases the response times due to over-utilizing the VMs.

[9] https://aws.amazon.com/ec2/pricing/.
[10] https://aws.amazon.com/lambda/pricing/.

Figure 9 compares METEL with the two native orchestration strategies implemented by Docker Swarm: Random and Spread. In the random strategy, the containers are placed randomly on the cluster nodes and the Spread strategy balances the cluster nodes by selecting the nodes with the least container load. As indicated in the figure, METEL outperforms both Random and Spread strategies.

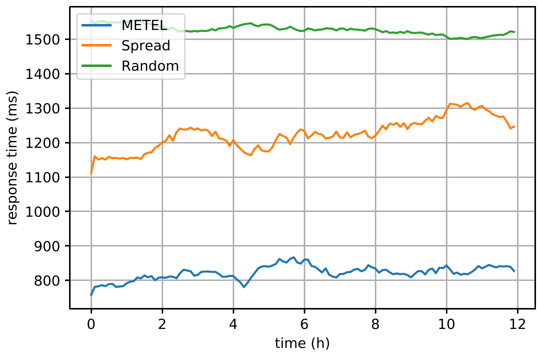


Fig. 9. Experimental comparison of METEL, Spread and Random orchestration strategies.

7 Conclusion

Although cloud computing can deliver scalable services for IoT network, the latency of data communication between ever increasing IoT devices and centralized cloud data centers can not be easily ignored and might be the bottleneck for many IoT applications. The edge-fog cloud promises to overcome the high amount of traffic generated by IoT devices, by bringing the cloud capabilities closer to the IoT endpoints.

In this paper, we introduced a minimizing end-to-end latency algorithm to provide the computation demand of IoT services in edge-fog cloud model. We also presented that serverless microservices provided by public clouds can be efficiently used in combination with in-house containerized services by our proposed mechanism. Moreover, our approach can be implemented as a complementary plugin in any container orchestration too.

In our experiments, we first showed that using traditional virtualization of resources is not scalable in an edge-fog cloud and containerization properly fits in such environments. Then, we analyzed how our application-aware scheduling mechanism can dramatically improve the utilization of fog resources to improve the response time, considering both proximity and compute capacity of edge, fog and cloud nodes. Finally, we observed that our results outperform the builtin Spread and Random scheduling mechanisms of Docker Swarm.

Considering the mobility of IoT nodes, migration of services is a major need. Although the lightweight containerization technology seems to be a proper choice for resource provisioning, its efficiency and applicability needs to be evaluated as future work.

Acknowledgement. This research has been funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation under Grant Agreement No. 785907 (Human Brain Project SGA2).

References

1. Aazam, M., Huh, E.N.: Dynamic resource provisioning through fog micro datacenter. In: IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), pp. 105–110, March 2015. https://doi.org/10.1109/PERCOMW.2015.7134002
2. Ahmed, A., Pierre, G.: Docker container deployment in fog computing infrastructures. In: IEEE International Conference on Edge Computing (EDGE), pp. 1–8, July 2018. https://doi.org/10.1109/EDGE.2018.00008
3. Bittencourt, L.F., Diaz-Montes, J., Buyya, R., Rana, O.F., Parashar, M.: Mobility-aware application scheduling in fog computing. IEEE Cloud Comput. **4**(2), 26–35 (2017). https://doi.org/10.1109/MCC.2017.27
4. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, pp. 13–16. ACM, New York (2012). https://doi.org/10.1145/2342509.2342513
5. Dua, R., Raja, A.R., Kakadia, D.: Virtualization vs. containerization to support PaaS. In: Proceedings of the 2014 IEEE International Conference on Cloud Engineering, IC2E 2014, Washington, DC, USA, pp. 610–614. IEEE Computer Society (2014). https://doi.org/10.1109/IC2E.2014.41
6. Fard, H.M., Prodan, R., Moser, G., Fahringer, T.: A bi-criteria truthful mechanism for scheduling of workflows in clouds. In: IEEE Third International Conference on Cloud Computing Technology and Science, pp. 599–605, November 2011. https://doi.org/10.1109/CloudCom.2011.92
7. Fard, H.M., Ristov, S., Prodan, R.: Handling the uncertainty in resource performance for executing workflow applications in clouds. In: IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC), pp. 89–98, December 2016. https://doi.org/10.1145/2996890.2996902
8. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and linux containers. In: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 171–172, March 2015. https://doi.org/10.1109/ISPASS.2015.7095802
9. Gupta, H., Dastjerdi, A.V., Ghosh, S.K., Buyya, R.: ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. Softw. Pract. Exp. (SPE) **47**(9), 1275–1296 (2017)
10. Kimovski, D., Ijaz, H., Surabh, N., Prodan, R.: Adaptive nature-inspired fog architecture. In: IEEE 2nd International Conference on Fog and Edge Computing (ICFEC), pp. 1–8, May 2018. https://doi.org/10.1109/CFEC.2018.8358723

11. Masip-Bruin, X., Marín-Tordera, E., Tashakor, G., Jukan, A., Ren, G.J.: Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems. IEEE Wirel. Commun. **23**(5), 120–128 (2016). https://doi.org/10.1109/MWC.2016.7721750
12. Morabito, R., Cozzolino, V., Ding, A.Y., Beijar, N., Ott, J.: Consolidate IoT edge computing with lightweight virtualization. IEEE Network **32**(1), 102–111 (2018). https://doi.org/10.1109/MNET.2018.1700175
13. Pahl, C., Lee, B.: Containers and clusters for edge cloud architectures - a technology review. In: 3rd International Conference on Future Internet of Things and Cloud, pp. 379–386, August 2015. https://doi.org/10.1109/FiCloud.2015.35
14. Pham, X.Q., Huh, E.N.: Towards task scheduling in a cloud-fog computing system. In: 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), pp. 1–4, October 2016
15. Scoca, V., Aral, A., Brandic, I., Nicola, R.D., Uriarte, R.B.: Scheduling latency-sensitive applications in edge computing. In: CLOSER (2018)
16. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016). https://doi.org/10.1109/JIOT.2016.2579198
17. Skarlat, O., Schulte, S., Borkowski, M., Leitner, P.: Resource provisioning for IoT services in the fog. In: IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), pp. 32–39, November 2016. https://doi.org/10.1109/SOCA.2016.10
18. Tasiopoulos, A., Ascigil, O., Psaras, I., Toumpis, S., Pavlou, G.: Fogspot: spot pricing for application provisioning in edge/fog computing. IEEE Trans. Serv. Comput. 1 (2019). https://doi.org/10.1109/TSC.2019.2895037
19. Villari, M., Fazio, M., Dustdar, S., Rana, O., Ranjan, R.: Osmotic computing: a new paradigm for edge/cloud integration. IEEE Cloud Comput. **3**(6), 76–83 (2016). https://doi.org/10.1109/MCC.2016.124
20. Yin, L., Luo, J., Luo, H.: Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. IEEE Trans. Ind. Inf. **14**(10), 4712–4721 (2018). https://doi.org/10.1109/TII.2018.2851241
21. Yousefpour, A., Ishigaki, G., Jue, J.P.: Fog computing: Towards minimizing delay in the internet of things. In: IEEE International Conference on Edge Computing (EDGE), pp. 17–24, June 2017. https://doi.org/10.1109/IEEE.EDGE.2017.12