This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2018.2851241, IEEE Transactions on Industrial Informatics

Tasks Scheduling and Resource Allocation in Fog Computing Based on Containers for Smart Manufacturing

Luxiu Yin, *Student Membership, IEEE*, Juan Luo*, *Membership, IEEE*, and Haibo Luo

***Abstract*—Fog computing has been proposed as an extension of cloud computing to provide computation, storage and network service in network edge. For smart manufacturing, fog computing can provide a wealth of computational and storage services, such as fault detection and state analysis of devices in assembly lines, if the middle layer between the industrial cloud and terminal device is considered. However, limited resources and low delay services hinder the application of new virtualization technologies in the task scheduling and resource management of fog computing. Thus, we build a new task scheduling model by considering the role of containers. Then, we construct a task scheduling algorithm to ensure that tasks are completed on time and the number of concurrent tasks for the fog node is optimized. Finally, we propose a reallocation mechanism to reduce task delays in accordance with the characteristics of the containers. Results showed that our proposed task scheduling algorithm and reallocation scheme can effectively reduce task delays and improve the concurrency number of tasks in fog nodes.**

***Index Terms*—Container, Fog computing, Resource management, Smart industry, Task scheduling.**

I. Introduction

A. Background

CLOUD computing is widely applied in many fields. However, many problems on cloud computing have emerged with the development of the so-called internet of things (IoT). Cloud computing cannot fully conform with delay-sensitive and location-aware applications considering the many types of IoT devices on the one hand, and cloud computing applications cannot be built extensively due to high construction costs on the other hand. With the increasing number of IoT devices, massive amounts of data are expected to be transferred to datacenters for processing in the coming years (i.e., Cisco predicts that connected devices will reach 50 billion and annual global data center IP traffic will reach 15.3 ZB by 2020 [1]). If IoT still uses the current cloud computing paradigm to handle the enormous amount of devices and data, it will give rise to high delay and network congestion. Based on these issues, fog computing was proposed by Cisco in 2012 [2] and defined by [3]. Fog computing is considered as extension of cloud computing to fix defects of cloud computing. In fog computing, the computation and storage will be migrated from the core of a network to the network edge to support real-time applications and reduce the network burden of datacenters.

Manuscript received January 30, 2018; revised May 04, 2018; accepted June 21, 2018. This work was supported by the National Natural Science Foundation of China (61672220), Key scientific and technological research and development plan of Hunan Province(2017GK2030), Electronic Information and Control of Fujian University Engineering Research Center, Minjiang University (EIC1701).

*Corresponding Author.

Luxiu Yin and Juan Luo are with College of Computer Science and Electronic Engineering, University of Hunan, Changsha, Hunan, P.R. China (e-mail: yinluxiu@hnu.edu.com, juanluo@hnu.edu.cn).

Haibo Luo is with Electronic Information and Control of Fujian University Engineering Research Center, Minjiang University(e-mail: 178573944@qq.com).

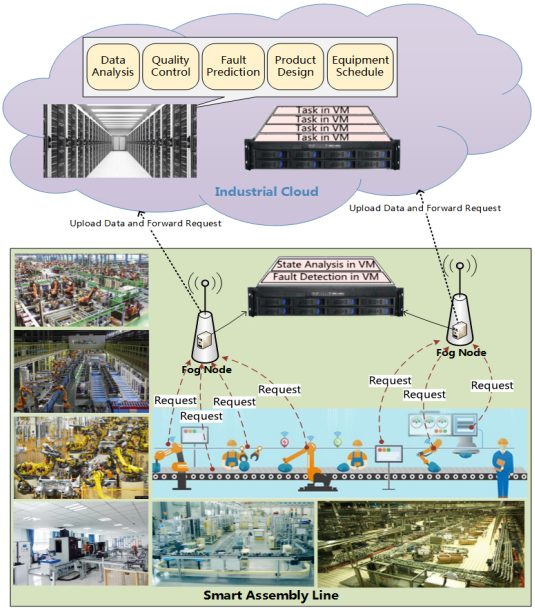


Fig. 1. The Scenario of Smart Assembly Line.

B. Motivation

A factory implementing smart manufacturing is equipped with several automatic assembly lines. As shown in the Fig. 1, this assembly line can include many devices, such as a robotic arm or a conveyor belt, to process each step of the production. In those devices, sensors collect real-time device status information, which are then used to detect exceptions and diagnose device faults. However, existing assembly devices exhibit poor computational capacity; sometimes, computational capacity is even lacking. Intensive computation tasks, such as real-time state analysis and fault detection, need to be delivered to high-performing servers to obtain the production response time. In automated assembly line production, several devices work together. If one of those devices failed, then the entire production process is derailed. For example, without immediate response, a failure of the robotic arm can lead to product flaws or even stop assembly line production. To reduce the impact of device failures, each device should upload its own sensing data to a high-performing server. The applications, which are run on a server, analyze real-time data and

immediately formulate appropriate decisions, such as enabling a backup equipment or notifying technicians for checking and repair. In the above mentioned case, if a fog node can immediately predict the failure (i.e., the sensing data uploaded by the robotic arm are analyzed in real time) and the relevant maintenance personnel is promptly notified, then the derailment to the production process may be avoided.

One of the major issues in smart manufacturing that requires urgent solutions is the analysis of real-time data and its corresponding feedback mechanisms. Fog computing provides computation, storage, and network resources to devices that lack computing resources in smart manufacturing and thus can solve real-time problems. Various devices upload their sensing data to nearby fog nodes, and these fog nodes process sensing data and respond to analytical results. However, the computation, storage, and network resources of existing fog nodes are limited. Thus, researchers have focused on how the resources on these fog nodes can be effectively utilized. Virtualization technology, a tool to effectively improve resource utilization, is applied to fog computing. In addition, when applied to fog nodes, virtualization technology not only improves resource utilization, but also solves other problems. Considering that fog nodes can provide resources to various devices and that each device requires its own application to process the sensed data, virtualization can create an isolated environment for each device to avoid the effects of resource competition. In existing virtualization technologies, virtual machines (VMs) are widely employed for many practical applications. However, using VMs is too complicated for tasks that need to be completed immediately. Fog computing is therefore proposed to provide faster response and serve more devices compared with VMs. Even if VMs require only a few seconds to start up, any delay is unacceptable for delay-sensitive tasks. Moreover, despite the superior isolation capability of VMs, the cost of VM monitoring is too high. Thus, VM performance decreases significantly when the number of VMs increases.

To overcome the known deficiencies of VMs, a lightweight virtualization technology called container has been widely used. Essential differences exist between containers and VMs. First, the isolation mechanism of a container differs from that of a VM. Container isolation is achieved through cgroups (Control Groups) rather than the hypervisor used by VMs. Cgroups is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes. A control group is a collection of processes that are bound by the same criteria and associated with a set of parameters or limits. These groups can be hierarchical, meaning that each group inherits limits from its parent group. The kernel provides access to multiple controllers (also called subsystems) through the cgroup interface. Containers can modify resources even if they are running, and this approach allows for the dynamic adjustment of resource usage on the basis of concurrent tasks. Second, containers and image files are strongly related. The creation of containers depends on their image files, whereas VMs are created without image files. The advantages of using containers and VMs are as follows:

- Containers start faster than VMs because hypervisors are not required by containers. The startup time of containers is expressed in milliseconds whereas that of VMs is expressed in seconds.
- Containers are better than VMs in terms of performance. IBM conducted a performance test of VMs and containers [4], and experiments showed that containers are superior to VMs in terms of CPU, memory, and I/O performance. The network performance of VMs is weaker than that of containers, but the difference is only by tens of microseconds.
- The greater the number of VMs deployed on a server, the higher the performance degradation of the server. By contrast, containers can be deployed in a server by the thousands, but the impact will not be significant.

C. Contributions

The current work on fog computing (e.g., resource management and task scheduling) are aimed for VM applications, even if containers have been regarded as a virtualization technology that is more lightweight and more efficient than VMs. Compared to containers, VMs require extremely high boot time and installation time for the running environment, which is fatal for delay-sensitive applications. For example, when a request which will be executed on a fog node, the fog node needs to create a virtual machine and then install the necessary running environment. Of course, the fog node can keep a virtual machine active that once a request arrives, this VM can immediately service it. However, this way will give rise to resources wasting when there are no requests to access the VM. And for task scheduling and resource allocation, virtual machines do not need to consider the placement and transmission of image files. However, many deficiencies in fog computing have been reported in the literature. Thus, we model a fog computing system on the basis of container characteristics and propose algorithms that can optimize the response time of tasks. Our main contributions are summarized as follows:

1) We aim to smart manufacturing and investigate task scheduling problems by employing containers in fog computing. Moreover, we model container-based task processing and consider the particularities of using containers in task processing.
2) We propose container-based task scheduling algorithms for the delay-sensitive and high-concurrency characteristics of fog computing. We divide the task execution processing into two sub-steps: determining if the task is either accepted or rejected and scheduling the accepted tasks to either the fog node or the cloud. In the proposed algorithms, all accepted tasks are considered completed within the delay constraint.
3) Unlike in VMs, modifying the resource quota of containers does not introduce additional overhead. Therefore, we propose a resource reallocation mechanism for the resource quota of each task, in which the fog node is expected to achieve resource utilization maximization. The reallocation mechanism is also expected to significantly reduce task delays.

The remainder of the paper is organized as follows. Section 2 summarizes the related work. Section 3 introduces our system model. Section 4 presents the task scheduling and resource management problems. Section 5 discusses the conducted experiments, the results of which verifies the effectiveness of our proposal. Finally, Section 6 concludes our work.

II. Related work

Fog computing is a popular research topic, and numerous studies have focused on its conception, architecture, and resource management aspect. In terms of conception, Bonomi et al. [2]

who proposed the fog computing paradigm, stressed that fog computing is a highly virtualized platform that typically, but not exclusively, provides computation, storage, and network services between end devices and traditional cloud computing datacenters at network edges. On the basis of previous work [3], Vaquero et al. [5] defined fog computing as a scenario wherein several heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralized devices communicate and potentially cooperate, after which a network performs storage and processing tasks without third-party intervention. Tasks can support basic network functions or new services and applications running in a sandboxed environment, and users who lease part of their devices to host these services receive incentives for doing so. In accordance with those definitions, several studies [3], [6] proposed application scenarios, such as smart home, connected vehicle, smart grid, wireless sensor, and actuators networks, on fog computing. In addition, [7], [8] highlighted the issues and challenges that need to be addressed, such as virtualization technology selection, security and privacy issues, and network management.

On the basis of the definitions and application scenarios, researchers focused on designing fog computing architectures. Xiang et al. [9] proposed a novel approach to mobile edge computing (MEC) for the IoT architecture called EdgeIoT, which can handle mobile data streams. Each base station was connected to a fog node that can provide local computing resources. The SDN-based cellular core was also designed to facilitate packet forwarding among fog nodes. A hierarchical fog computing architecture in each fog node was proposed to provide flexible IoT services while maintaining user privacy. Hou et al. [10] combined Hypertext Transfer Protocol and Message Queuing Telemetry Transport servers to provide IoT services in the IoT cloud architecture, which in essence was similar to fog computing. Several components, such as application servers, database clusters, brokers, and load balancers, were employed to guarantee high system performance. Zhang et al. [11] constructed a network architecture with a three-layer model, in which the fog nodes, massive datacenter operators (MDCOs), and data service subscribers (DSSs) were designated in the upper, middle, and bottom layers, respectively. A hierarchical game was also proposed. The interaction between fog nodes and MDCOs was regarded as a multi-leader multi-follower Stackelberg game, and the interaction between MDCOs and DSSs was regarded a single-leader single-follower Stackelberg game. Given that decision making was distributed, the fog nodes, MDCOs, and DSSs were effectively utilized. Bo et al. [12] proposed a hierarchical distributed fog computing architecture to support the integration of massive number of infrastructure components and service in future smart cities. The authors enhance the "smartness" of city infrastructure by employing advanced machine learning algorithms across all system layers.

At present, there have been many researches [13]–[15] on resource management and task scheduling for cloud computing. Fog computing resource management and task scheduling are at the initial phase. However, Zeng et al. [16] who focused on a fog computing-supported software-defined embedded system, investigated the following problem statements: 1) how to balance the workload on a client device and computation servers, i.e., task scheduling; 2) how to place task images on storage servers, i.e., resource management; and 3) how to balance the I/O interruption requests among storage servers. Mao et al. [17], [18] developed an online joint radio and computational resource management algorithm for multi-user MEC systems, and the objective was to minimize the long-term average weighted sum power consumption of mobile devices and MEC servers, subject to a task buffer stability constraint. Furthermore, a green MEC system with energy harvesting devices was investigated, an effective computation offloading strategy was developed, and a low-complexity online algorithm (i.e., Lyapunov optimization-based dynamic computation offloading algorithm) was proposed. The algorithm could select offloading decisions for CPU cycle frequencies during mobile execution and power transmission for offloading computation. Ma et al. [19] proposed a cloud assisted mobile edge computing (CAME) framework to enhance system computing capacity and a workload scheduling mechanism to balance the tradeoff between system delay and cost. Xiao et al. [20] proposed a novel cooperation strategy referred to as offload forwarding. In this scheme, instead of constantly relying on cloud datacenters for processing, each fog node could forward part or all of its unprocessed workload to its neighboring fog nodes to further improve QoE of users.

Containers are regarded as an efficient and lightweight virtualization technology and thus have been recently investigated [4], [21]. For cloud service providers such as Amazon and Aliyun, containers are also provided to users as a basic service. In industrial applications, an increasing number of open-source projects focused on containers. For instance, OpenStack integrated containers after the release of Liberty. Flynn [22] and Deis [23] are also container-based PaaS projects.

To summarize the related works, we find that existing task scheduling and resource management in fog computing generally involved VMs or bare metal models. Most studies ignored the high concurrency of fog nodes, and these studies focused mainly on how to reduce delays but did not consider how to ensure task completion given a certain delay constraint. In this paper, we propose a container-based task scheduling model and task scheduling algorithms with a task delay constraint. Furthermore, in accordance with the characteristics of the container, a resource reallocation mechanism is proposed to reduce the execution delay of tasks.

III. System Model

The fog computing system is modeled as Fig. 2, with set N of terminal devices in the assembly line with poor computation capacity and set J of fog nodes for the high-performing servers around those devices. An industrial cloud located far from the devices offers various complex cloud services (e.g., quality control, product design, and equipment scheduling) for the terminal devices and provide computation, storage, and network resources for the fog nodes with insufficient resources. The terminal devices can access the fog node through a wireless network (e.g., WiFi, Bluetooth, and LoRa) to which they submit their requests. Each fog node comprises three components for task scheduling and resource management: request evaluator, task scheduler, and resource manager. The request evaluator is responsible for determining, accepting, or rejecting tasks. The task scheduler distributes tasks to either the cloud or the fog node. The resource manager reallocates the resources of each task running in the fog node.

We consider the fog computing system to be a hard real-time system, which means that an acceptable request must be accomplished before the deadline specified by its terminal device. If the fog node is unable to accomplish the request on time,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2018.2851241, IEEE Transactions on Industrial Informatics

it means that the fog node does not have enough resources to allocate to the request. The fog node should reject the request and inform the terminal device to resubmit the new request within the new deadline. The request evaluator determines requests according to the order of arrival of requests. For example, a terminal device submits a request for fault detection. The request evaluator determines whether the fog node or cloud can satisfy the delay constraint of the request. If the request is refused, then the request evaluator sends a message to inform the terminal device to resubmit the request with a new delay. If the request is accepted, then the request evaluator transfers the request to the task scheduler. The task scheduler resolves the request by task and determines if the task is run on the fog or the cloud. If the task is designated to the cloud, then the task scheduler delivers the task to the cloud. If the task is designated to the fog, then task scheduler delivers the task to the resource manager. The resource manager receives the task and allocates the corresponding resource to the task. The task is considered as the minimum unit and thus cannot be divided into multiple subtasks on different servers. Once a task is run on the fog node or the cloud, it exists as a container. Finally, the resource manager reallocates the resource to each task, which is activated in the fog node.

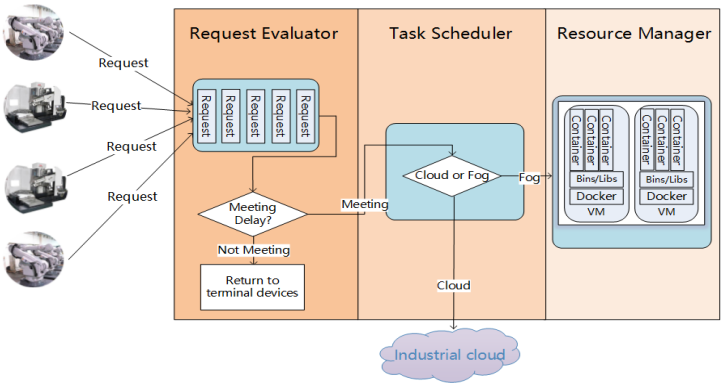


Fig. 2. System Model

We denote T as a set of tasks requested by all terminal devices. Each task $t \in T$ is characterized by a three-tuple of parameters, $t \triangleq \langle Data, Delay, Image \rangle$. The $Data$ parameter processing data, in which the unit of data is KB. The $Delay$ parameter represents the deadline of the task t which is generated by the terminal device. We assume that delay value as rational, and the unit of the delay is seconds. The $Image$ parameter refers to a Uniform Resource Identifier (URI) in the store instead of an image file and thus can reduce the size of requests. Each fog node has a set I^j of image files, which are used frequently within their own store because it is unable to store all image files within a finite resource. The image files must be uploaded to the store before the fog nodes or industrial cloud can create the corresponding container. In our system, we assume that industrial cloud has unlimited resources and thus can image store reproduction in DockerHub [24].

A. Fog Execution Model

If the task t is distributed to the fog node, then the fog node determines whether the task can be run locally, or transmitted to the cloud or rejected, based on its execution time. The execution time of a task in a fog node depends on the computation time, data, and image transmission time denoted by $\tau^j_{t,com}$, $\tau^j_{t,data}$, and $\tau^j_{t,img}$, respectively. We assume that the output of the computation is small so the transmission delay for feedback is negligible. Hence, the execution time in fog node j of task t can be calculated as

$$Ex^j_t = \tau^j_{t,com} + \tau^j_{t,data} + \tau^j_{t,img}. \tag{1}$$

In Eq.(1), similar to those presented in previous work [20], [25], [26], some differences are observed when containers are considered. The computation time of task t in the fog node j $\tau^j_{t,com}$ is calculated as

$$\tau^j_{t,com} = \frac{v_{t,data}}{\sum_{p=1}^{P^t_{max}} r^j_{t,p}}, \tag{2}$$

where $r^j_{t,p}$ represents the allocated computational resources by fog node j for the task t during the period p, while P^t_{max} denotes the maximum number of period that the task can maintain. To satisfy the delay constraint, P^t_{max} can be calculated by:

$$P^t_{max} = \lfloor Delay_t / \rho_j \rfloor, \tag{3}$$

where ρ_j is a constant, which is expressed as the time of each period.

The resource amount needed by task t is calculated as

$$Re(t) = \varepsilon^j_t \cdot v_{t,data}, \tag{4}$$

where $v_{t,data}$ represents the data volume need to be processed by task t, while ε^j_t denotes the number of instructions to process the per bit data of images needed by task t. The unit is MIPS and it can be derived from the offline measurement [27].

To ensure that the task can be completed on time, the task must be allocated with more resources than it needs, such that

$$Re(t) \leq \sum_{p=1}^{P^t} r^j_{t,p}, \tag{5}$$

where P^t is the actual number of periods consumed by the task t.

The data transmission time of task t can be calculated as:

$$\tau^j_{t,data} = \frac{v_{t,data}}{w^j_n}, \tag{6}$$

where w^j_n represents the bandwidth allocated by fog node j.

The image transmission time of the task t is

$$\tau^j_{t,img} = \begin{cases} 0, & Image_t \in I^j \\ \frac{v_{t,img}}{w^{j,c}_t} & Image_t \notin I^j, \end{cases} \tag{7}$$

where $v_{t,img}$ denotes the image volume needed by task t, while $w^{j,c}_t$ represents the bandwidth that fog node j allocates to the task t to connect it to the cloud.

The following constraint is employed in the fog execution model:

$$\sum_{t \in T^j_p} r^j_{t,p} \leq R^j, \tag{8}$$

where T^j_p represents the task set which in the fog node j during the period p. The constraint indicates that the resources occupied by the tasks running on the fog node cannot exceed the total resources of the node in any period.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2018.2851241, IEEE Transactions on Industrial Informatics

B. Cloud Execution Model

In the cloud execution model, we assume that the computational resources of the cloud is infinite. Therefore, when a task arrives, it can be executed immediately without waiting for computational resources. In addition, all image files are stored in the industrial cloud. Hence, image transmission time can be ignored. The execution time of a task t that runs on the cloud can be calculated as

$$Ex_t^c = \tau_{t,com}^c + \tau_{t,data}^j + \tau_{t,data}^{j,c}, \tag{9}$$

where $\tau_{t,com}^c$ represents the computation time of task t in the industrial cloud. In accordance with our computational model, if the cloud can provide enough resources, then the computation time of its tasks in the cloud is negligible[1]. Furthermore, $\tau_{t,data}^{j,c}$ denotes the data transmission time of task t from fog node j to the industrial cloud. Considering that the computational resource of the cloud is assumed to be infinite, all image files can be stored without considering the transmission time.

The data transmission time from fog node j to cloud $\tau_{t,data}^{j,c}$ of task t can be calculated by:

$$\tau_{t,data}^{j,c} = \frac{v_{t,data}}{w_t^{j,c}}. \tag{10}$$

In addition, the delay constraint also needs to be satisfied, such that

$$x_t^j Ex_t^j + x_t^c Ex_t^c \leq Delay_t, \tag{11}$$

where x_t^j and x_t^c are binary variables to indicate whether the task t is running in fog node j or the cloud, respectively.If the task is running in the fog node or the cloud, then the value is 1; otherwise, 0. On the basis of the indivisible assumption of the task, $x_t^j \neq x_t^c$.

IV. Problem Formulation

This section divides the task scheduling and resource management problems of fog computing into three sub-problem statements on the basis of container characteristics.

1) When a request is sent to a fog node, how to estimate whether the request can be completed within the delay constraint, and subsequently, how to decide whether to accept the request;
2) After the request is accepted, as a means to improve the efficiency of the fog node, how to determine whether the request is executed on the local fog node or forwarded to the cloud;
3) On the basis of container isolation mechanism, how to design the corresponding resource reallocation mechanism to maximize the resource utilization of the fog node.

A. Request Evaluation

Fog nodes need to maximize their resource utilization, which means fog nodes need to accept as many tasks as possible. However, this requirement does not mean that all requests will be accepted. For example, if the fog node accepts several tasks in a period, which can result in insufficient resources available to these tasks, the delay constraint of the request will not be satisfied. Therefore, an incoming request should only be accepted if the delay constraint of the request is satisfied by the fog node or the cloud. The delay constraint of the request, which is run on the cloud, can be easily calculated by obtaining the radio resource w_n^j of terminal device n, fog node j and bandwidth $w_t^{j,c}$ allocated for task t between fog node j and the industrial cloud. However, the execution time of the fog node cannot be easily calculated because the computation time of a task depends on the resource allocated by the fog node. The resource quota of the assigned task is determined by the resource manager that decides in accordance with the current task of running in the fog node, but this can lead to a failure in predicting the completion time of the request. In addition, the resource manager dynamically reallocates the resource to become efficient. Therefore, we convert the calculation of computation time in the fog node into the calculation of available resource. The fog node can then determine whether the delay constraint of the task can be met, i.e., by calculating whether sufficient resources are available for a certain time.

When a request is sent to the fog node, the total amount of remaining resources is first calculated for all tasks that need to be completed before the period point p_{max}^t. We use $\widetilde{T}_{p_{max}^t}^j$ to represent the task set, which needs to be completed before the period point p_{max}^t, P'^t is the number of periods that the task t has run; hence, the amount of remaining resource of active tasks in the fog node j can be calculated as

$$\sum_{t \in \widetilde{T}_{p_{max}^t}^j} Re_{rem}^j(t) = \sum_{t \in \widetilde{T}_{p_{max}^t}^j} (Re(t) - \sum_{p=1}^{P'^t} r_{t,p}^j). \tag{12}$$

For the tasks to be completed after p_{max}^t, the request evaluator calculates the minimum resource requirements for those tasks before p_{max}^t. This approach can determine whether task t can obtain sufficient resources during the $Delay_t$. Therefore, we assume that the resource manager will not allocate resource to those task before period point p_{max}^t. Then, we calculate the difference $\triangle R_p^j$ between the amount of remaining resources of the task set $\widetilde{T'}_{p_{max}^t}^j$ that the tasks must be completed after p_{max}^t and the resource limitation of the fog node j in each period,

$$\triangle R_p^j = \sum_{t \in \widetilde{T'}_{p_{max}^t}^j} Re_{rem}^j(t) - R^j. \tag{13}$$

If the remaining resource exceeds the resource limit of the fog node, i.e., $\triangle R_p^j > 0$, it means that these resources must be allocated before the period point p_{max}^t. If remaining resource of the task t does not exceed the resource limit of the fog node j, i.e., $\triangle R_p^j \leq 0$, it means that these resources will be accumulated into the calculation of the subsequent period, as Eq(14).

$$\triangle R_{p+1}^j = \sum_{t \in \widetilde{T'}_{p_{max}^t}^j} Re_{rem}^j(t) - R^j - \left|\triangle R_p^j\right|. \tag{14}$$

The minimum resource requirement before p_{max}^t of those tasks that need to be completed after p_{max}^t is given by Eq.(15):

$$\sum_{p=p_{max}^t+1}^{p_{last}^j} \triangle R_p^j, \triangle R_p^j > 0, \tag{15}$$

[1]In the real environment, the computation time in the cloud needs to be considered. However, the fog node cannot confirm the computation time of tasks in the cloud. Therefore, when the fog node accepts a task, it needs to forward the task information to the cloud at the same time. Then, the cloud evaluates the computation time of the task and returns it to the fog node. Lastly, the fog node makes a second evaluation of the task.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2018.2851241, IEEE Transactions on Industrial Informatics

where p^j_{last} represents the largest end period point of the currently active tasks. Then the request evaluator can make the decision by

$$(p^t_{max}-p+1)R^j+R^j_{p,rem}\geq \sum_{p=p^t_{max}+1}^{p^j_{last}} \triangle R^j_p + \sum_{t\in \widetilde{T}^j_{p^t_{max}}} Re^j_{rem}(t)+Re(t), \triangle R^j_p>0, \tag{16}$$

where $R^j_{p,rem}$ denotes the remaining resource in period point p of fog node j.

The evaluation algorithm is shown in the Algorithm 1. The lines 4-9 are used to find tasks which can be completed after p^t_{max} from the current active task set and calculate the resources amount of these tasks. This part needs to traversal all active tasks. Therefore, the complexity of this part is $O(T^j_p)$. Then, in line 10, we calculate the required resources amount before p^t_{max} of tasks which will be completed after p^t_{max}. To achieve this goal, we first traverse all the period point from to , and accumulate the total amount of resources that must be required in each period. Therefore, the complexity of this part is $O((p^j_{last}-p^t_{max})\times \widetilde{T}'^j_{p^t_{max}})$.

Algorithm 1 Request Evaluation Algorithm

Input: Request t;
Output: boolean result;
1: Calculate $t.amountRes$ using Eq.(4);
2: Set $p^j_{last}=0, needRes=0, \widetilde{T}^j_{p'^t_{max}}=\{\}$;
3: Find the max period point of current active tasks and set p^j_{last};
4: **for** task i in T^j_p **do**
5: **if** $i.p^t_{max}>t.p^t_{max}$ **then**
6: Put i into $\widetilde{T}^j_{p'^t_{max}}$;
7: Calculate requirement resource $needRes$ before p^t_{max} using Eq.(12);
8: **end if**
9: **end for**
10: Calculate the minimum resource requirement before p^t_{max} of those tasks that need to be completed after p^t_{max};
11: **if** $t.amountRes+needRes<(p^t_{max}-p+1)R^j+R^j_{p,rem}$ **then**
12: return true;
13: **else**
14: return false;
15: **end if**

B. Task Scheduling

The task scheduler distributes the task when its request is accepted. If the task can only be completed in either the cloud or the fog node, then it is directly distributed. If the cloud and the fog node can both complete the task, then the task scheduler needs to decide where to place the task. Fog nodes assist the cloud in accomplishing tasks that are less computationally intensive. By contrast, tasks that are highly computationally expensive (i.e., overly resourced) are forwarded to the cloud. For example, if a task requires too much resources, then the fog node cannot provide sufficient resource for the upcoming task. The task therefore needs to upload the processed data to the cloud. However, even if the data that need to be processed by these tasks are limited, too much time is spent for data transmission. The fundamental purpose of fog computing is to exchange computational resources with network resources to solve the problem of insufficient resources of terminal devices. Considering that constructing a function to express the relationship between computational and network resources is difficult, we designed a threshold and adjusted it in accordance with the current set of tasks to maximize the number of active tasks on the fog node. However, this threshold is also difficult to determine.

The task scheduler maximizes the number of active tasks on fog node j at each period and this approach is regarded as the optimization objective of task scheduling. However, because it is online problem, the tasks are scheduled one by one; hence, the task scheduler cannot obtain all the tasks and perform optimal scheduling. Therefore, the task scheduler needs to set the decision threshold δ^j_p for each period, i.e., resource threshold of fog node j in the period p. When the task is accepted, the task scheduler first calculates the resource demand $Re^j_{avg}(t)$ per period of the task, which is calculated as

$$Re^j_{avg}(t)=\frac{Re(t)}{P^t_{max}}. \tag{17}$$

And then, the task scheduler compares $Re^j_{avg}(t)$ with the threshold δ^j_p of current period, if the $Re^j_{avg}(t)>\delta^j_p$, then the task should be distributed to the cloud. If $Re^j_{avg}(t)\leq\delta^j_p$, the task will be distributed to the fog node for execution. In this paper, we calculate the resource requirements per period rather than focusing on the total amount of resources because if a task has a large amount of resources with long delay constraint (i.e., the average in each period of the resource demand is not large), then the impact on the node density will not be high. However, for a short delay constraint, the cloud should be scheduled because the cloud handles several resources relative to the fog nodes. Scheduling such tasks on the cloud can greatly reduce computing time.

Instead of always using a fixed threshold, the task scheduler adjusts δ^j_p in each period on the basis of the number of accepted tasks in the last period. We assume that tasks are continually sent to the task scheduler. Therefore, if the number of tasks accepted in the current period is less than the last period, then too many tasks are allocated to the cloud due to the threshold value of the current period being too small. Thus, the tasks that exceed the threshold acceptable by the fog node are rejected. If the number of tasks accepted in the current period is greater than the number of tasks in the previous period, then the threshold of the current period may not be the optimal value. The task scheduler continues to decrease the threshold in accordance with the average of all tasks in the current period, and this can be expressed as

$$\delta^j_{p+1}=\begin{cases}\delta^j_{p-1}, & p>1, T^j_p<T^j_{p-1}\\ max(Re^j_{avg}(t)|t\in T^j_p), & p>1, T^j_p\geq T^j_{p-1}\\ R^j, & p=1\end{cases}. \tag{18}$$

The threshold update algorithm is shown in the Algorithm 2. In this algorithm, we only need to find the maximum average resource demand of current active tasks or update to last threshold. Therefore, the complexity of the threshold update algorithm is $O(T^j_p)$.

C. Resource Reallocation

With the container isolation mechanism, the resource quota of a container can be modified immediately without having to wait for the container to restart or wait for any delay. We develop the optimal reallocation scheme for the resource manager, which can calculate the resource quota of each task in the subsequent period

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2018.2851241, IEEE Transactions on Industrial Informatics

Algorithm 2 The Threshold Update Algorithm

Input: $\delta_p^j, receivedRecordList$;
Output: δ_{p+1}^j;
1: Calculate the amount of received tasks in current period;
2: Get the amount of received tasks in last period from receivedRecordList;
3: $receivedTasks \leftarrow$ the amount of received tasks in current period;
4: $lastReceivedTasks \leftarrow$ the amount of received tasks in last period;
5: **if** $receivedTasks \geq lastReceivedTask$ **then**
6: Calculate the average per period resource quote of each task;
7: Calculate δ_{p+1}^j using Eq.(18);
8: **end if**
9: return δ_{p+1}^j;

during the execution phase. We maximized data processing per cycle by reallocating the CPU resources in the fog node.

In our system model, if a task is distributed to the fog node, then the resource manager immediately allocates resources to the task instead of waiting for the subsequent period. By the end of the period, the resource manager determines the tasks that currently run on the fog node and reallocates resources to them. In accordance with our assumption, the resources of the cloud is unlimited, and hence, we mainly focus on the reallocation in the fog node.

$$\begin{aligned} \text{min:} \quad & -\sum_{t\in T_p^j} r_{t,p}^j/\varepsilon_p^j \\ \text{s. t.:} \quad & \sum_{t\in T_p^j} r_{t,p}^j \leq R^j \\ & r_{t,rem}^j \leq r_{t,p}^j, \forall t \in \{t|p_{max}^t = p^j\} \end{aligned} \tag{19}$$

The second constraint implies that the tasks that need to be completed in this period must be allocated enough resources. The objective function Eq. (19) is a linear programming problem, and thus standard linear programming algorithms, such as the interior point method, are utilized. By our test, only a few milliseconds in needed to solve a task set with a size of 500.

Furthermore, the resource manager is responsible for the initialization of the task during resource allocation. When a task is forwarded to the resource manager, the resource manager needs to promptly allocate resources to the task. Given that the request evaluator simply determines whether the fog node has enough resources assigned to the task, the resource manager needs to ensure that the task acquires enough resources within its delay constraint when performing resource allocation initialization. Initial resource allocation involves the following situations:

1) The max end period point of the task is equal to the current period point, i.e., $p_{max}^t = p^j$. In this situation, the resource manager searches the tasks that can be delayed without violating the delay constraints and reduce the resource quota for the incoming task.
2) The max end period point of the task is greater than the current period point, i.e., $p_{max}^t \geq p^j + 1$. For those tasks, the resource manager first allocates the average resource and then reallocates the other in the remaining period.

V. Simulation Experiments

We initially compare the performance of VMs and containers for the most significant characteristic of fog computing, i.e., concurrent condition. We consider containers, which are the foundation of our model, to be more suitable than VMs for fog computing. To prove this, we run multiple Linpack programs simultaneously to test the performance loss of multi-program parallel runtime.

A. Containers vs. VMs

We use a server with 16 GB of memory and an E5-2620v4@2.10 GHz (a total of 8 cores and 16 threads) processor as the fog node, the operating system of the server is ubuntu16.04. In addition, we used Docker 1.12.6, QEMU 2.5.0 and Libvirt 1.3.1 for our experiments.

Linpack [28] is used to test the performance of containers and VMs under the concurrency condition. In this experiment, GNOME is employed to simultaneously start multiple terminals and simulate the concurrency. We set up all containers and VMs to share the physical resources. For example, the resources of a container will be 16 GB of memory with an eight-core CPU when only one container or VM exists. If two containers or VMs exist, each container or VM will have 4 GB of memory with a four-core CPU. Ubuntu 16.04 for the container base images, which is also the operating system of VMs. The array size of Linpack is 200*200. Each level repeats ten experiments to calculate the average value of the result and to avoid result occasionality. Fig. 3 shows that the performance of containers/VMs decreases when their amount is increased. Containers and VMs decrease at an equal rate from 1 to 3. By contrast, from 4 to 10, the decrease of the containers is slower than that of the VMs. Therefore, in the case of high concurrency, the CPU performance of containers is better than that of VMs. Due to the performance limitations of the server, when we create 11 VMs, the complete result is not fully obtained because of some VM failures.

In addition, we developed a simulation system, which is based on Java with multithreading. A request queue, which is used to receive requests from terminal devices, is added to the system. Our system employed multithreading technology to simulate the sending processes of terminal devices and execution process of each task. Each task and terminal device is designated as a thread. The terminal device sends requests periodically, and the data transmission time is simulated by sleeping, which corresponds to the thread of terminal device. When the request is distributed to the fog node, the fog node create a task object as a thread. The computation delay of tasks also corresponds to thread sleeping. The thread is stopped when the task is completed.

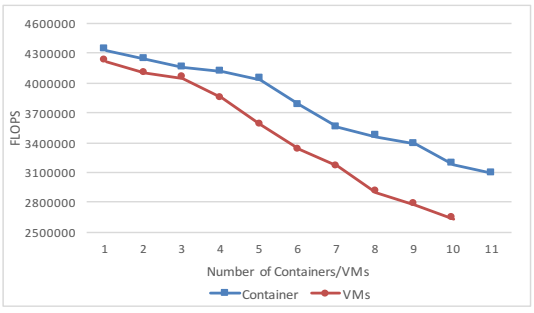


Fig. 3. FLOPS (Floating-point Operations Per Second) of Containers/VMs

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2018.2851241, IEEE Transactions on Industrial Informatics

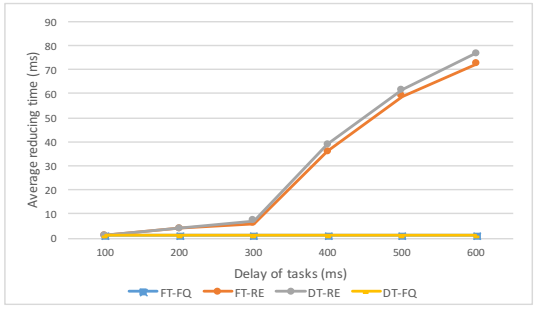


Fig. 4. Average reducing time of accepted tasks

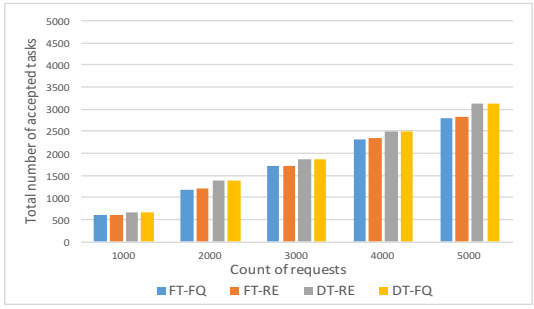
B. Task Scheduling and Resource Management

In our simulation, the configuration of the system is set with a schedule cycle of 100 ms. Ten users concurrently send requests. We generated five datasets with the random algorithm, and the sizes of the datasets are 1,000, 2,000, 3,000, 4,000, and 5,000. The data of each request ranged from 1,000 to 10,000 kb. The processing rate of the images is derived from a previous work [27]. However, in accordance with another research [27] that used CPU cycle as the unit of data processing, we converted CPU cycle into MIPS. The delay of tasks ranged from 100 to 600 ms in accordance with the data volume.

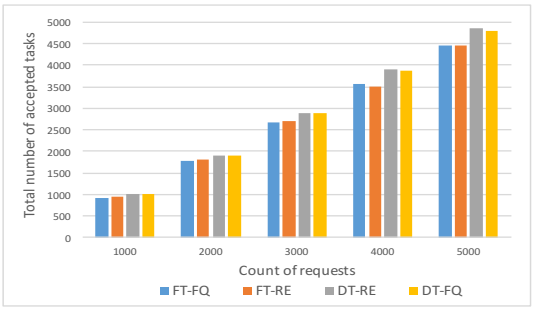
The compared task scheduling algorithms are as follows:

1) Fixed threshold + Fixed quota(FT-FQ): FT-FQ uses fixed thresholds and fixed resource quota. Because the task scheduling is a online problem, thus scheduling tasks by a fixed threshold is the most direct and simplest way. In addition, because the most of researches [17], [25], [26] did not focuses on containers, once the resources have been allocated to a task, its resource quota will no longer be modified.
2) Fixed threshold + Reallocation(FT-RE): FT-RE uses fixed thresholds for task scheduling and reallocation mechanism for resource management proposed by this paper. This approach takes advantage of containers without considering the high concurrency of fog nodes.
3) Dynamic threshold + Fixed quota(DT-FQ): DT-FQ uses dynamic threshold for task scheduling, which proposed by this paper and fixed resource quotas for resource management. This approach considers the high concurrency of fog nodes but does not leverage the characteristics of containers.
4) Dynamic threshold + Reallocation(DT-RE): DT-RE uses dynamic threshold and reallocation mechanism, which all proposed by this paper. This approach combines the high concurrency of tasks with the characteristics of containers.

Fig. 4 presents the result of average reduction in terms of delay. Then, we compared the different datasets. According to the results, FT-RE and DT-RE that used reallocation mechanism significantly reduced the delay when the tasks delay constraint range from 300 to 600 ms, while the delay of tasks are in the range from 100 to 200 ms, the delay of reducing is lower. This is due to the fact that the resource quota per period is optimized as the reallocated number of a task increases. The optimization goal of the reallocation mechanism is to improve the resource utilization of the fog node, which means increasing the data processing capacity of the node. And because the amount volume of data for each task is fixed, increasing the fog node's data processing capacity means reducing the task's completion time. However, for those tasks with short delay constraints, the task delay can not be



(a) 10 MIPS



(b) 20 MIPS

Fig. 5. Total number of accepted tasks

significantly reduced due to the small number of reallocations or even none. Regarding as FT-FQ and DT-FQ, the delay of tasks does not be reduced because the resource manager allocates resources based on each task's $Re^j_{avg}(t)$ in order to ensure that the task can be completed on time.

In Fig. 5(a) the dynamic threshold increased the number of task concurrency by 10 MIPS. In Fig. 5(b) the dynamic threshold increased the number of task concurrency by 20 MIPS. The results suggest that DT-FQ and DT-RE, which employ the dynamic threshold, can increase the number of accepted tasks. Due to the lack of ability to learn and adjust in terms of situation of requests, the approach of fixed threshold causes that most of tasks will be distributed to the fog node and quickly run out of its resources in the first few periods, leaving the upcoming tasks with insufficient resources to be assigned while many of those tasks can be completed using fewer resources. However, the approach of dynamic threshold can constantly adjust the threshold to find the best value for the current situation of requests. By updating threshold, some tasks that require a lot of resources will be distributed to the cloud, and thus the fog node can save the resources for those tasks that require smaller resources.

VI. Conclusion

In this paper, we first introduced the application scenario of fog computing in smart manufacturing, the terminal devices of the assembly line need real-time data analysis for detecting exceptions and diagnosing device faults. When computation tasks are offloaded, the fog node and the terminal devices, which currently cannot sufficiently compute data processing, are ultimately provided with high-quality computation. Based on these problems, we designated containers as the task unit to provide the computational resources to the terminal devices in the section III. Then, on the basis of container characteristics, we modeled the task scheduling process, proposed the novel task scheduling algorithm, and designed the resource reallocation scheme to improve the

resource utilization of fog nodes and reduce task delays. Lastly, our experiments showed that the scheduling algorithm can increase 5% the number of accepted tasks and the reallocation mechanism can significantly decrease about 10% execute time for each tasks.

However, to simplify the service model, we ignore the finiteness of cloud resources and then omit the computation time in the cloud. In real situation, this factor should be taken into account. In addition, the image placement of containers is a significant problem, we will work hard to solve this problem in the future in order to further reduce task execution time and network traffic.

REFERENCES

[1] Cisco, "Cisco global cloud index: Forecast and methodology, 2015–2020," *Cisco*, 2015.

[2] F. Bonomi, R. Milito, and J. Zhu, "Fog computing and its role in the internet of things," *Mcc Workshop on Mobile Cloud Computing ACM*, pp. 13–16, Aug 2012.

[3] F. Bonomi, R. A. Milito, P. Natarajan *et al.*, "Fog computing: A platform for internet of things and analytics," *Big Data and Internet of Things: A Roadmap for Smart Environments*, pp. 169–186, 2011.

[4] W. Felter, A. Ferreira, R. Rajamony *et al.*, "An updated performance comparison of virtual machines and linux containers," March 2015.

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

[6] Z. Li, B. Chang, S. Wang *et al.*, "Dynamic compressive wide-band spectrum sensing based on channel energy reconstruction in cognitive internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, January 2018.

[7] M. Mukherjee, R. Matam, L. Shu *et al.*, "Security and privacy in fog computing: Challenges," *IEEE Access*, vol. 5, no. 19293-19304, Sept 2017.

[8] Y. Wang, T. Uehara, and R. Sasaki, "Fog computing: Issues and challenges in security and forensics." Taichung, Taiwan: IEEE Computer Software and Applications Conference, July 2015.

[9] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the internet of things," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 22–29, Dec 2016.

[10] L. Hou, S. Zhao, X. Xiong *et al.*, "Internet of things cloud: Architecture and implementation," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 32–39, Dec 2016.

[11] H. Zhang, Y. Zhang, Y. Gu *et al.*, "A hierarchical game framework for resource management in fog computing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 52–57, 2017.

[12] B. Tang, Z. Chen, G. Hefferman *et al.*, "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Transactions on Industrial Infomatics*, vol. 13, no. 5, pp. 2140–2150, Oct 2017.

[13] J. Nie, J. Luo, and L. Yin, "Energy-aware multidimensional resource allocation algorithm in cloud data center." *KSII Transactions on Internet and Information systems*, vol. 11, no. 9, pp. 4165–4187, 2017.

[14] K. Xie, J. Cao, X. Wang *et al.*, "Optimal resource allocation for reliable and energy efficient cooperative communications." *IEEE Transactions on Wireless Communications*, vol. 12, no. 10, pp. 4994–5007, 2013.

[15] B. Yang, Z. Li, S. Chen *et al.*, "Stackelberg game approach for energy-aware resource allocation in data center." *IEEE Transactions on Parallel and Distributed Computing*, vol. 27, no. 12, pp. 3646–3658, 2016.

[16] D. Zeng, L. Gu, S. Guo *et al.*, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, Dec 2016.

[17] Y. Mao, J. Zhang, S. Song *et al.*, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, Sept 2017.

[21] K. Kaur, T. Dhand, N. Kumar *et al.*, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 48–56, 2017.

[18] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, Dec 2016.

[19] X. Ma, S. Zhang, W. Li *et al.*, "Cost-efficient workload scheduling in cloud assisted mobile edge computing." Vilanova i la Geltru, Spain: IEEE/ACM International Symposium on Quality of Service, 2017, pp. 1–10.

[20] Y. Xiao and M. Krunz, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation." Atlanta, GA, USA: IEEE Conference on Computer Communications, May 2017.

[22] "Flynn," https://flynn.io/, Accessed 31 January 2018.

[23] "Deis," https://deis.com/, Accessed 31 January 2018.

[24] "Dockerhub," https://hub.docker.com/, Accessed 31 January 2018.

[25] A.-C. Pang, W.-H. Chung, T.-C. Chiu *et al.*, "Latency-driven cooperative task computing in multi-user fog-radio access networks." Atlanta, GA, USA: IEEE 37th International Conference on Distributed Computing Systems, June 2017.

[26] X. Meng, W. Wang, and Z. Zhang, "Delay-constrained hybrid computation offloading with cloud and fog computing," *IEEE Access*, vol. 5, no. 21355-21367, 2017.

[27] A. Miettinen and J. Nurminen, "Energy efficiency of mobile clients in cloud computing," USENIX Association Berkeley. Boston: HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, June 2010, pp. 4–4.

[28] "Linpack," https://github.com/ereyes01/linpack, Accessed 31 January 2018.

Luxiu Yin received the Bachelor's degree in Computer Science and Technology in 2011 from Changsha University, Hunan, China. He achieved Master's degree in Computer software and theory in 2015 from Hunan Normal University. He is currently receiving for a Ph.D. degree in college of information science and engineering, Hunan University, Hunan, China. His research is focused on cloud computing, fog computing and wireless virtual network.

Juan Luo (M'10) received the Bachelor's degree in 1997 from National University of Defense Technology, Hunan, China. She achieved Master's and Ph.D. degrees in communication and information system in 2000 and 2005, respectively, both from Wuhan University, Hubei,China. She is currently a professor and doctoral supervisor at the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, China. From 2008 to 2009, she was a visiting scholar in the University of California at Irvine. She has published more than 70 papers. Her research is focused on internet of things, cloud computing and middleware. She is a member of the IEEE and SIGCOM, a member of the ACM, and a senior member of CCF.

Haibo Luo received the Bachelor's degree in Communication Engineering from Wuhan University of Technology, China, in 2006. the Master's degree in Information and Communication Engineering from Hunan University, China, in 2009. He achieved the Ph.D. degree in College of Physics and Information Engineering, Fuzhou University, Fuzhou, China, in 2018. His research interests include Internet of Things, fog computing and mobile computing.