Accepted Manuscript

Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT

Juan Luo, Luxiu Yin, Jinyu Hu, Chun Wang, Xuan Liu, Xin Fan, Haibo Luo

PII: S0167-739X(18)31358-X
DOI: https://doi.org/10.1016/j.future.2018.12.063
Reference: FUTURE 4684

To appear in: *Future Generation Computer Systems*

Received date : 1 June 2018
Revised date : 8 October 2018
Accepted date : 29 December 2018

Please cite this article as: J. Luo, L. Yin, J. Hu et al., Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT, *Future Generation Computer Systems* (2019), https://doi.org/10.1016/j.future.2018.12.063

Container-based Fog Computing Architecture and Energy-balancing Scheduling Algorithm for Energy IoT

Juan Luo[a], Luxiu Yin[a], Jinyu Hu[a], Chun Wang[a], Xuan Liu[a,*], Xin Fan[a], Haibo Luo[b]

[a] *College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, China*
[b] *Electronic Information and Control of Fujian University Engineering Research Center, Minjiang University, Fujian, China*

Abstract

The traditional architecture of fog computing is for one data center and multiple fog nodes. It is unable to adapt to the current development of private cloud. In addition, virtual machines used for cloud computing, are also used for fog computing as the resource unit, can not satisfy the requirement of fog computing. Furthermore, the limited capacity of battery power has been one of the major constraints when considering cloud-to-fog-to-sensor pattern in the scenario of Energy Internet. We propose a multi-cloud to multi-fog architecture and design two kinds of service models by employing containers to improve the resource utilization of fog nodes and reduce the service delay. According to the two service models, we present a task scheduling algorithm for energy balancing. The algorithm is based on the transmission energy consumption of terminal devices and uses a dynamic threshold strategy to schedule requests in real time, thereby guaranteeing the energy balancing of terminal devices without increasing the transmission delay. Experimental results show that our proposed service models and scheduling algorithm can reduce service latency, improve fog node efficiency, and prolong WSNs life cycle through energy balancing.

Keywords: Container, Docker, Energy Balancing, Fog computing, Multi-cloud.

*Corresponding author

Email addresses: juanluo@hnu.edu.cn (Juan Luo), yinluxiu@hnu.edu.cn (Luxiu Yin), jinyuhu@hnu.edu.cn (Jinyu Hu), cwang@hnu.edu.cn (Chun Wang), xuanliu1022@gmail.com (Xuan Liu), Xinfan@hnu.edu.cn (Xin Fan), 178573944@qq.com (Haibo Luo)

1. Introduction

1.1. Background

Cloud computing has been applied widely in many fields. However, many problems with cloud computing have emerged with the development of IoT. In front of a mass of IoT devices, cloud computing has difficulty satisfying delay-sensitive and location-aware applications because it cannot be built extensively due to the expensive cost of construction. With the number of IoT devices growing (Cisco predicts that there will be 50 billion connected devices by 2020 [1]), an ocean of data will be transferred to data centers for processing (annual global data center IP traffic will reach 15.3 ZB by the end of 2020). If IoT still uses the current cloud computing paradigm to handle the enormous amount of devices and data, it will give rise to high latency and network congestion. Based on these issues, fog computing was proposed by Cisco in 2012 [2], and defined by L. M. et al [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 data centers.

1.2. Motivation

Although there is extensive research on fog computing, there are still several issues. First, most of the resource management of fog computing is based on virtual machines (VMs). However, using VMs as the resource is not adaptable to fog computing platforms. For example, the boot-up time of a VM is several minutes, which is too long for real-time applications. For a large number of access points, the characteristics of fog computing are affected by the number of VMs because the performance of physical machines is degraded when the number of VMs increases. The overhead of a hypervisor exponentially increases when more VMs run within the same machine [4]. Therefore, a new resource form is required to support the features of fog computing. Second, the conventional IaaS platform is not adapted to fog computing. The service paradigm of conventional IaaS is that users send their demands for virtual resources to the data center, and the data center subsequently creates the VM in terms of the demand and returns an IP address to the user. Users manage the VM by accessing the IP, and last, the user installs various binaries and libraries to lay the groundwork for whatever application needs to run. For example, if an application uses Elasticsearch, then the VM has to install Java and Lucene before deploying Elasticsearch. This procedure is tedious and inappropriate for the real-time response to fog computing. Third, as shown in Fig. 1, most fog computing researches [5, 6, 7] are aimed at one data center with multiple fog nodes. However, increasingly, enterprises are building their own private data centers in order to store and analyze data. At the same time, the enterprises need to deploy their applications on fog nodes to collect data and provide real-time response to terminal devices. Despite the low cost of building a single fog node, it is necessary to deploy a host of fog nodes to cover a wide range. The construction and maintenance cost of these fog nodes is tremendous for enterprises, especially for those enterprises whose services only run for a short term or periodically.

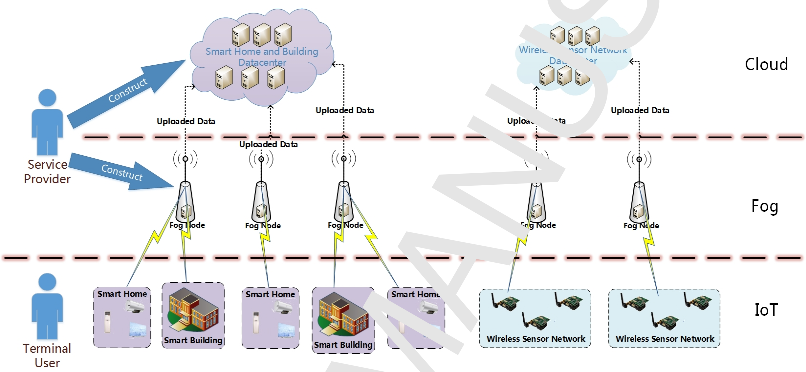


Figure 1: Traditional architecture of fog computing

As an improved version of the IoT, Energy Internet (EI) integrates energy and information in a distributed and renewable energy system, which has been well studied by numerous researchers[8, 9, 10, 11] in recent years. And when considering cloud-to-fog-to-sensor pattern in the scenario of Energy Internet, energy efficiency of sensor node has still been the most crucial issue on fog computing. Because of the renewable nature of EI, energy management is an essential part on fog computing, which has significant effects on energy balancing among sensor nodes. In this paper, our objective is to balance the distribution of residual energy of the overall network nodes. Therefore, one of the greatest challenges in energy management is to select an optimal way to balance the energy consumption on offloading the tasks from energy-limited sensors to energy-sufficient clouds/fogs.

1.3. Contributions

Aiming to solve the problems mentioned above, we propose a novel hierarchical architecture that is called multi-cloud to multi-fog architecture, and design a temporary service model and a long-term service model. Furthermore, we propose a task scheduling algorithm for energy balancing. The main contributions of this paper are described as following:

- We propose a novel architecture and design several components to solve potential security issues caused by multi-cloud to multi-fog as well as to reduce the construction and maintenance cost of service developers.
- We design two service models, a temporary service model and a long-term service model, based on containers which are used as the resource units. The temporary service model provides real-time request and response services, while the long-term service model is responsible for subscription and publishing service. Our experiments show that the service models based on containers can effectively improve the resource utilization of fog nodes and reduce service delays.
- We propose a task scheduling algorithm based on energy balancing to prolong the life of WSNs and reducing the delay constraint of tasks. In the scheduling algorithm, we design an energy balancing strategy to control the transmission power of terminal devices and use a dynamic threshold to schedule arrived requests in real time. The algorithm improves the number of concurrent tasks in the fog node and balances the energy consumption of terminal devices without increasing the data transmission time.

The rest of this paper is structured as follows. Section 2 is the related work. Section 3 details the architecture and presents the implementation of components in this architecture. Section 4 elaborates the task scheduling algorithm for energy balancing. The experimental results of containers and VMs are evaluated and discussed in Section 5. Section 6 summarizes our study and outlines our future research.

2. Related Work

At present, the researches on the fog computing architecture are mainly focus on the architecture, resource management, task scheduling of fog computing.

Sun et al. [12] first proposed an SDN-based network architecture. The network architecture places the SDN in the middle of the fog node and the cloud data center, which improves the communication efficiency between the fog node and the cloud. In addition, the article also gives the fog node service architecture. Its service architecture is divided into two layers, application layer and user layer. At the user data layer, a user's private virtual machine is deployed, the original data sent by the user is processed, and the processed data is sent to the application-level service virtual machine. Through the processing of the original data, the user's privacy is guaranteed. Lu Hou et al. [13] combined HTTP and MQTT to implement a request response model and a message subscription and release model, respectively, to satisfy the service diversity of fog computing. In addition, the implementation of the message broker is given and a three-level QoS level is designed to improve service reliability. The architecture uses several components such as application servers, database clusters and agents, load balancers and so forth to ensure its high performance.

Mao et al. [14, 15] 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. [16] 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.

Container-based virtualization presents an interesting alternative to VMs in the cloud. In the cloud, containers have been proposed as a lightweight virtualized technology. There are several studies comparing the performance of containers and VMs [17, 18]. We contacted these studies to discuss which virtualization technology is more suitable to fog computing for deployment convenience, performance.

Deployment convenience: Using containers, everything required to make a piece of software run is packaged into isolated containers. Unlike VMs, containers do not bundle a full operating system, and only libraries and settings required to make the software work are needed. This approach makes for efficient, lightweight, self-contained systems and guarantees that software will always run the same, regardless of where it is deployed. This property makes the deployment of a container simpler and easier than a VM and hides the heterogeneity of fog computing. Furthermore, due to the service object of fog computing being mobile, live migration has to be considered. To date, the container is weak for live migration because of nested process migration. Although the live migration of containers can be implemented by CRIU [19], the nested relationship of process groups among multiple containers is not sustainable. However, for a single container, live migration is supported well. In comparison to containers, the live migration of VMs has developed maturely. At present, there are solutions based on memory [20] and disk [21]. Moreover, it is easy to execute without nested process problems since VMs are migrated with an integrated object.

Performance: The container has approximately native performance compared to the virtual machine because the container can run on the operating system directly, while the virtual machine has a hypervisor layer. IBM undertook the research [17] comparing the performance between virtual machines and Linux containers. In the aspects of memory and CPU, both virtual machines and containers introduce negligible overhead. For I/O, containers are much better than virtual machines when they perform intensive I/O operations. Regarding network latency, despite container latency being double the native, compared to virtual machines, it is a negligible difference (only a difference of 10μs-20μs). This paper also compares the service performance including Redis and Mysql. The results show that the latency and throughput of a container are similar to native, while the virtual machine is inferior to the container. Overall, containers are better than virtual machines for the real-time characteristic of fog computing.

3. The Architecture of Fog Computing

3.1. Roles

Different from service developers and infrastructure providers as one role in conventional fog computing, we divide it into two roles. In conventional fog computing, a service developer not only develops service but also constructs fog nodes in an extensive range. However, the cost of constructing fog nodes in an extensive range is expensive for service developers if their services are running just for few months; therefore, renting to the resource of fog nodes to run their services can efficiently reduce the cost to the service developer. There are three roles in our architecture, infrastructure provider (InP), service developer (SD) and terminal device (TD). InPs construct fog nodes for service developers and terminal devices, service developers develop services that can run in the fog nodes, terminal devices use the service via submitting a request to a fog node and paying for it.

3.2. Multi-cloud to Multi-fog Architecture

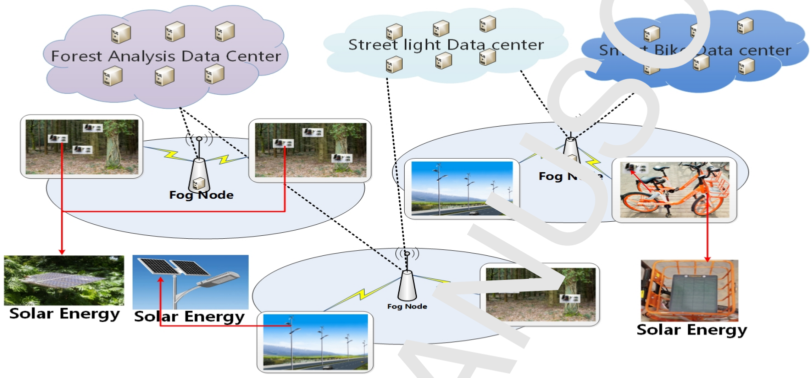


Figure 2: Multi-cloud to multi-fog architecture

In this paper, we propose a multi-cloud to multi-fog architecture based on containers. The architecture is designed with a hierarchy using containers as the resource unit. In our architecture, a fog node consists of three tiers, that is, the infrastructure tier, the control tier and the access tier. The infrastructure tier virtualizes the physical resource and offers the API of managing a container to the control tier. The control tier orchestrates services for enhancing resource utilization and provides the interaction between fog nodes and SDs. The access tier is used for request management and access control. A request is processed from the top tier to the lower tier. When the fog node receives a request, first, the request will be validated for permission and resolved into the service catalog at the access tier. Next, the service catalog is transferred to the control tier. The control tier dispatches the services in the service catalog to the corresponding manager, either temporary or long-term according to the service type. Last, the corresponding manager calls the API provided by the infrastructure tier to create the container. As shown in the Fig. 2, there are three SDs and three fog nodes. Each fog node provisions the physical resource for TDs near the node location, as well as SDs. There are several services running in fog nodes; these fog nodes connect to multiple private clouds of SD with the Internet. The services are encapsulated to image-file by the SD, and the image-file is actively pushed to fog nodes by the SD or downloaded by users to fog nodes. The fog node starts the services by running image-files. TDs upload data to the fog node for computing and storing. The data that must be uploaded to the cloud is packaged by the fog node and regularly transferred to the cloud.

3.3. Components of Multi-cloud to Multi-fog architecture based on Containers

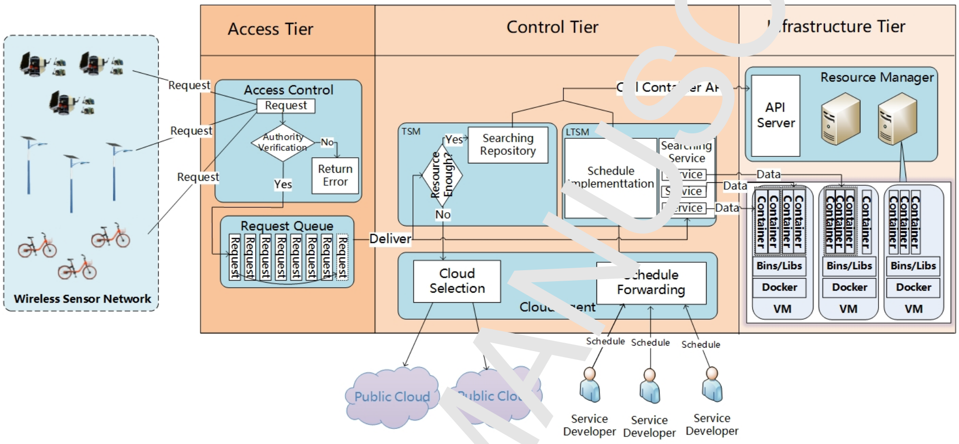


Figure 3: Components of multi-cloud to multi-fog architecture based on container

The lower tier is the infrastructure tier. In the infrastructure tier, the object to be managed is the container. The physical resources are provided by one or more physical machines. These PMs have installed hypervisor, which means that these PMs will be segmented to a number of VMs. However, as mentioned earlier, using a VM as the unit of resource is not efficient. For temporary service, a container is a better alternative because the container has absolute superiority relative to VMs as a resource unit in fog computing. Containers take up less disk space. A container is represented by lightweight image-files. VMs are also based on images but fuller monolithic ones than a container. Note that rather than deploying a container directly on a physical machine, we use the VM as the host of containers because the incomplete isolation of a container, which shares the system kernel can cause insecurity. We are unable to create a VM for each container since the performance of the physical machine will decrease with the number of VMs increasing. Consequently, containers-on-VM is adopted to alleviate the damage from a system failure caused by a container so that even a container breakout or system failure only affects the current VM, reducing the influence of the risk of containers. In summary, the resource manager should be introduced in this tier. The resource manager provides a managed interface of containers to the above tier and offers the managed interface of VMs for InPs. The resource manager is described as follows:

Resource Manager: In the resource manager, we choose Docker as the container engine and KVM (Kernel-based Virtual Machine) as the hypervisor. Docker is the most popular container engine. It uses Docker Container as the basic unit for resource segmentation and scheduling, encapsulates the entire software runtime environment and is designed for developers to build, publish and run distributed applications. The core of Docker is Docker daemon; it will start the API Server that provides interfaces, including container creating, updating, deleting and searching to receive requests from the Docker client (the client refers to those components of the above tier). QEMU-KVM is employed as a hypervisor. KVM only supports the CPU and memory virtualization; therefore, it needs to combine QEMU to provide a complete virtualization. In addition, InPs manage virtual machines via Libvirt that is the most widely used tool and application programming interface (API) for managing KVM virtual machines. Several commonly used virtual machine management tools, such as virsh, virt-install, virt-manager and the cloud computing framework are using it at the foundation.

The middle tier is control tier. In this tier, the object to be managed is service. In general, long-term services rent the resource of a fog node until the contract expires. Temporary services are submitted with the data processed by the service. When the service is finished, the resource that is used by the service will be released. Therefore, because of the difference between long-term service and temporary service, there are two managed components, Long-term Service Manager (LTSM) and Temporary Service Manager (TSM), for the life cycle of containers of long-term service and temporary service.

LTSM: Long-term services are deployed by SD on the fog node that is constructed by InP. Thus, a terminal user is unaware of whether the service has been deployed in the fog node and does not know the port of the service when the user submits their requests. The service registration and discovery must to be leveraged by the LTSM. LTSMs manage the service in their own fog node, and collaborate with the LTSM of neighbor nodes. Since the resources in a fog node are finite, the service will be migrated to other fog nodes or suspended when resources are not sufficient. For example, if all neighbor nodes of a fog node host a service that is accessed infrequently and occupies considerable resource in this fog node, then the fog node should suspend the service and release resources and subsequently forward those requests that access this service to neighbor nodes with current fog node ID. When a service is accessed frequently by this fog node, the fog node should restart the service to reduce the overhead of communication. However, which service should be migrated or suspended and how to select the forwarding destination are still open issues. InP can design various heuristic schedules in terms of an InP's demand such as minimizing communication cost or optimizing performance.

TSM: Temporary services are more sensitive than long-term services for the latency of communication, and most temporary services are applied by TDs that are mobile. Therefore, in order to minimize latency, the service should be migrated when the terminal user roams out of the coverage of the fog node. The TSM of the current fog node notices the TSM of the destination fog node in advance according to the path of the TD to ensure a seamless hand over and take over. Because the size of some image-files is too large relative to a TD from a few KB to a few GB, TSM has a database for storing image-files, which are used frequently. For example, the ubuntu image-file requested by many users as a base image should be stored in the fog node, the fog node creates the container directly instead of downloading the image-file from an image-file repository such as DockHub to alleviate the cost of communication between the fog node and image-file warehouse. For an image-file that is not stored in the repository, users just submit the URL of the image-file in the repository, and then the fog node will automatically download this image and run it. The manager uses a database to store the MD5 of image-files. The image manager periodically deletes image-files, which are accessed infrequently.

Cloud Agent: The cloud agent plays an important role in this architecture. The cloud agent builds a bridge between fog nodes and clouds to improve efficiency and protect the system. We design the cloud agent based on both long-term service and temporary service.

For long-term service, the cloud agent offers two functionalities. The first is the service proxy. The long-term service should be under SD control, whereas a fog node is shared by a multitude of SDs. Considering the security of the fog node, an SD is unable to operate Docker directly. The only way to manage their services is by the cloud agent. The management includes creating, updating and deleting service, increasing or reducing the service resources. When the SD needs to manage its services, it will send the request to the cloud agent, then the cloud agent calls the API provided by the Resource Manager to respond to the request. For the long-term service that runs periodically, the SD sends its service schedule to the cloud agent and the cloud agent works with LTSM to implement the schedule. The cloud agent can avoid malicious behavior or incorrect operation that causes potential insecurity. The second is supervising. SD supervises long-term services; however, due to running conditions, such as the utilization of CPU, memory, I/O and the network of service that is offered by Docker in the infrastructure tier, the service is unable to perceive its own running condition. Therefore, the running condition is submitted to the cloud agent by Docker, and the cloud agent delivers it according to the service affiliation.

Temporary service will commonly be hosted in the fog node. However, the fog node serves a sea of users with limited resources in the fog node. If a single temporary service applies for excessive resources that will cause other users wait too long or not be satisfied by the fog node and thus affect the user experience, they should be migrated to clouds that registered their cloud service in the fog node. The cloud agent will periodically send heartbeat information to those clouds in order to detect the network delay for each registered cloud. When TSM receives the request, if the TSM finds that the request cannot be hosted in the fog node, then the TSM transmits the service request to the cloud agent. The cloud agent should select a cloud service provider, which minimizes the delay to reduce the latency.

The top tier is the access tier. The object to be managed is a request. The request is responsible for authorizing users, as well as receiving requests. The request uses JSON to transfer the message to hide the diversity of IoT devices. A request for temporary service consists of an image-file URL list, container number, to-be-processed data; the request that accesses long-term service consists of a unique identifier of service, to-be-processed data, and service token.

Request Queue: Due to fog computing needs to process high concurrency requests, which have different priorities. Fog nodes require a component to assign the request according to the priority of the request. Therefore, we choose the priority queue in fog computing, rather than a traditional FIFO queue. The request manager has a request queue that stores forthcoming requests and addresses the priority of these requests. If a request has high priority, then the request queue will deliver this request to LTSM/TSM priority.

Access Controller: The access controller is divided into network access control and service access control. The network access control is responsible for managing the terminal user connection to a fog node. The network access control uses the username and password to verify that the terminal user can connect to the fog node. The service access control is employed to manage which services the terminal user can access; it works primarily with long-term services. For service access control, the access controller has a table for storing the token of each service running in the fog node. The service token is provided by SD. The SD also provides the corresponding token to the terminal user with devices or applications, the terminal user submits the token with the request to the access controller. If the token in the request matches the token in the access controller, the access controller pushes the request to request queue, otherwise the access controller returns an error to terminal user.

4. Energy Balancing for Terminal Devices

4.1. System Model

The fog computing system is modeled as Fig. 3, with set N of TDs and a fog node for the high-performing servers around those devices. TDs will periodically collect the environment data and harvest energy. Because of the limitation of computation capacity of TDs, the collected data need to be transferred to the fog node for processing and analyzing. In addition, TDs will send the heart-beat messages regularly to the fog node which connect with those TDs. A heart-beat message includes the ID of the TD, the energy collected by the current time slot and the energy consumption and battery capacity at the current time slot. In each time slot, the fog node calculates the available transmitting power for each TD according to its own heart-beat messages. However, if the current battery capacity of the device does not meet the energy demand for data transmitting at the next slot, then it is determined that the energy consumption of the device is insufficient and the device is adjusted to a dormancy state, waiting for the energy harvesting equipment to be charged.

4.1.1. Energy Model of TDs

Because the fog node can support wired power supply, the main consideration is the energy model of TDs. A TD consists of several sensors and a energy harvesting equipment. In this paper, we use solar panel to harvest energy. The battery capacity of the TD in the time slot k can be calculated as follow:

$$E_i(k+1) = E_i(k) + E_{i,har}(t) - E_{i,cost}(k), \tag{1}$$

where $E_{i,cost}(k)$ and $E_{i,har}(k)$ denote the energy cost and the harvested energy of the TD i in time slot t respectively.

Since the TD cannot process the collected data within a very low delay, it needs to upload the collected data to the fog node for processing. Therefore, the energy cost of the TD can be obtained by :

$$E_{i,cost}(k) = power_{i,k} \times \frac{v_i(k)}{w_n^j \times log_2(1 + \frac{h_k \times power_i}{o})}, \tag{2}$$

where $power_{i,k}$ denotes the transmission power. The second item of the right side of the equation is the data transmission delay, $v_i(k)$ represents the collected data volume in the time slot k, w is system bandwidth, h_k is the channel power gain at the kth time slot, o is noise power at the TD.

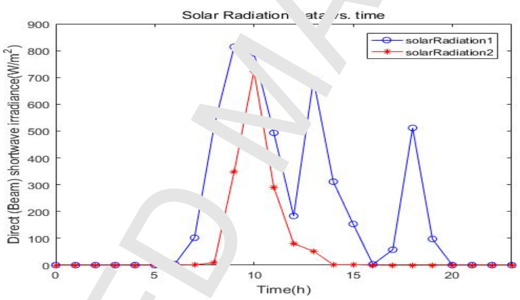


Figure 4: The harvested solar energy

The energy harvested by solar panels varies with time, season, and other factors. At 10 am, energy harvesting equipment is the most efficient. When entering the night, the energy collected by the collecting device is 0. From our experiment, we know that the energy harvested by solar panels at different time periods, as shown in Fig. 4. Therefore, in this paper, we will use the energy harvesting value in our following experiments.

4.1.2. Fog Execution Model

When a fog node receives a data sent by TDs, the fog node determines whether the task can run locally or transmitted to the cloud according to its execution time. Each task $t \in T$ is characterized by a three-tuple of parameters, $t \triangleq \langle Data, Delay, Image \rangle$, where the *Data* represents the processing data. The unit of data is KB. The *Delay* represents the deadline of the task t which is generated by the TD. The *Image* parameter refers to a Uniform Resource Identifier. The execution time of a task in a fog node depends on computation time denoted by $Ex^j_{com}(t)$ and its data and image transmission time denoted by $Ex^j_{tra}(t)$ and $Ex^j_{img}(t)$ respectively. We assume that the output of the computation is of small size so that the transmission delay for feedback is negligible. Hence, the execution time in the fog node j of task t can be calculated as

$$Ex^j(t) = Ex^j_{com}(t) + Ex^j_{img}(t) + Ex^j_{tra}(t). \tag{3}$$

In Eq.(3), similar to the [22, 23], but there are some differences due to use of containers. The computation time $Ex^j_{com}(t)$ of the task t in the fog node j can be calculated by:

$$Ex^j_{com}(t) = \frac{v_{t,data}}{\sum_{1\le p<P^t_{max}} f^j_p(t)}, \tag{4}$$

where $f^j_p(t)$ represents allocated computation resources by the fog node j for the task t during the period p. P^t_{max} represents the maximum number of period that the task can stay. In order to satisfy the delay constraint, P^t_{max} can be calculated by:

$$P^t_{max} = \lfloor \frac{D(t)}{\rho_j} \rfloor, \tag{5}$$

where ρ_j is a constant, which is expressed as the time of a period and $D(t)$ represents the delay of the task t.

The resource amount which task t need can be calculated by:

$$F(t) = \varepsilon^j_t \cdot v_{t,data} \tag{6}$$

$v_{t,data}$ represents the data volume need to be processed of the task t, ε^j_t represents the number of instructions needed to process per bit of data for image which needs by the task t. Its unit is MIPS and it can be measured [24].

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

$$Ex^j_{tra}(t) = \frac{v_i(k)}{w^j_n \times log_2(1 + \frac{h_t + power_{i,k}}{o})}, \tag{7}$$

The image transmission time of the task t is

$$Ex^j_{img}(t) = \begin{cases} 0, & Image_t \in I^j; \\ \frac{v_{t,img}}{w^c_j} & Image_t \notin I^j, \end{cases} \tag{8}$$

where $v_{t,img}$ represents the image volume needs the bandwidth used to connect to the cloud when the fog node j allocates to the task t.

In addition, there is a constraint in the fog execution model:

$$\sum_{t \in T_p^j} f_p^j(t) \leq F^j, \tag{9}$$

where T_p^j represents the task set which in the fog node j during the period p and F^j represents the total computation resource of the fog node j. The constraint indicates that the resources occupied by the tasks running on the fog node can not exceed the total resources of the node in any period.

4.1.3. Cloud Execution Model

In the cloud execution model, we assume that the computation resource of the cloud is infinite. Therefore, when a task arrival, it can be executed immediately without waiting for computation resource. In addition, the all image file will be stored in the cloud and hence the image transmission time can be ignored. The execution time of a task t that is run in the cloud can be calculated as

$$Ex^c(t) = Ex^j_{tra}(t) + Ex^{j,c}_{data}(t), \tag{10}$$

where $Ex^{j,c}_{data}(t)$ represents the data transmission time of the task t from the fog node j to the cloud. Because computation resource of cloud is assumed to be infinite, all image files can be stored without regard to the transfer time of the image files. According to our computational model, if the cloud can provide enough resources, the computing time of its tasks in the cloud will be negligible.

The data transmission time of task t which from the fog node j to cloud can be calculated by:

$$Ex^{j,c}_{data}(t) = \frac{v_{t,data}}{w_t^{j,c}}, \tag{11}$$

where $w_t^{j,c}$ represents the data transmission rate from the fog node j to cloud.

The delay constraint should be met, i.e.,

$$x_t^j Ex^j(t) + x_t^c Ex^c(t) \leq D(t), \tag{12}$$

x_t^j and x_t^c are binary variables, which indicate whether the task t is running on the fog node j and whether it is running in the cloud respectively. If running in the fog node or the cloud, their value is 1, otherwise 0. Based on the indivisible assumption of the task, and hence, $x_t^j \neq x_t^c$.

4.2. Algorithm Description

We need to balance the energy consumption of TDs within the entire network for increasing the life cycle of a wireless sensor network. If the energy of a TD is exhausted prematurely, the entire network life cycle will end prematurely. However, a TD cannot obtain the energy status information of other devices and thus cannot make decisions on transmission power. In order to ensure that the remaining battery capacity of TDs within the entire network can be maintained on the same level, all devices to periodically send their own energy status information to the fog node. Then the fog node plans the calculation power and transmission power of each TD. This ensures that the remaining battery capacity of each node can be kept at the same level. Note that all data must be completed within the specified delay.

In order to balance the energy consumption of each device, we need to limit the standard deviation of the energy consumption of each TD and the entire network energy consumption within a specific range.When the remaining battery capacity of a device is lower than the network average, we need to reduce its transmission power so that it can reduce the energy overhead. When the remaining battery capacity of a device is higher than the average value, we can increase its transmission power, thereby reducing the delay of data transmission. Since there is no guarantee that the energy consumption of all devices is exactly the same, we define an offset value θ that allows a certain amount of deviation. Consequently, energy balancing can be expressed as follow:

$$|E_i(k+1) - E_i(k)| \leq \frac{\sum_{j \in N} |E_j(k+1) - E_j(k)|}{N} + \theta, \tag{13}$$

then we bring it in Eq. (1) to obtain the energy consumption that the device i can use in the next time slot. According to the energy consumption in next time slot, we can calculate the transmission power via Eq. (2).

Since the energy balance of the WSN needs to be ensured, the transmission energy cost of the TD needs to be adjusted, resulting in an increase of the data transmission delay. For those delay-sensitive applications, delay-sensitive requirements need to be met by reducing the scheduling of fog nodes. If the uploaded data is large, the fog node needs to make a decision that upload the uploaded data locally or forward it to the cloud for processing to meet the delay of tasks.

Task scheduler in TSM will distribute the task when the task has arrived. Fog nodes are used to assist the cloud to accomplish those tasks that are less computationally intensive, so tasks that are overly computationally expensive, that is, overly resourced, need to be forwarded to the cloud. However, the data need to be processed of these tasks are very small, but spent a lot of time in data transmission. Since the task scheduler needs real-time scheduling, that is, whenever a task arrives, it is scheduled immediately. This approach makes the scheduler unable to obtain all tasks and perform optimal scheduling. Therefore, we design a threshold and adjust it according to the current set of tasks, to maximize the number of active tasks on the fog node. However, this threshold is difficult to determine.

Task scheduler maximizes the number of active tasks on a fog node j at each period as the optimization objectives for task scheduling. However, it is online problem, which means the tasks are scheduled one by one and hence the task scheduler cannot obtain all the tasks and perform the optimal scheduling. Therefore, the task scheduler needs to set a decision threshold δ_p^j for each period, indicating the resource threshold of the fog node j in the period p. When the task is accepted, the task scheduler first calculates the resource demand $F_{avg}^j(t)$ per period of the task, which is calculated as:

$$F_{avg}^j(t) = \frac{F(t)}{P_{max}^t} \tag{14}$$

And then compared with the threshold δ_p^j of current period, if the $F_{avg}^j(t) > \delta_p^j$, then the task shall be distributed to the cloud; otherwise, the task will be distributed to the fog node for execution. It should be noted that this paper calculates the resource requirements per period rather than the total amount of resources. Since it will not cause too much impact on the node density in the case of a task with a large amount of resources and longer delay constraint, which means that each period of resource demand is not too much on average. However, for a task with large amount of resources but a short delay constraint, the cloud can be scheduled because the cloud is extremely rich in resources relative to the fog nodes. Scheduling such tasks to the cloud can greatly reduce the computing time.

Instead of always using a fixed threshold, the task scheduler adjusts δ_p^j in each period based on the number of accepted tasks in last period. We assume that tasks will be continually sent to the task scheduler, therefore, if the number of tasks accepted in the current period is less than the previous period, it means that too many tasks are allocated to the cloud due to the threshold value of the current period being too small so that the tasks that can be accepted by the fog node exceed the threshold will be rejected. If the number of tasks accepted in the current period is greater than the number of tasks in the previous period, it means that the threshold of the current period may not realize the optimal value. The task scheduler continues to decrease the threshold according to the maximal resource quota of all active tasks in the current period,

$$\delta_{p+1}^{j} = \begin{cases} \delta_{p-1}^{j}, & p > 1, T_p^j < T_{p-1}^j \\ max(F_{avg}^{j}(t) | t \in T_p^j), & p > 1, T_p^j \geq T_{p-1}^j \\ F^j, & p = 1, \end{cases} \tag{15}$$

Algorithm 1 The Threshold Update Algorithm

Input: $\delta_p^j, receivedRecordList$;

Output: δ_{p+1}^j;

Calculate the amount of received tasks in current period;
Get the amount of received tasks in last period from receivedRecordList;
$T_p^j \leftarrow$ the amount of received tasks in current period;
$T_{p-1}^j \leftarrow$ the amount of received tasks in last period;
if $T_p^j \geq T_{p-1}^j$ **then**
    Calculate the average per period resource quote of each task;
    Calculate δ_{p+1}^j using Eq. (15);
else
    $\delta_{p+1}^j \leftarrow \delta_{p-1}^j$;
end if
return δ_{p+1}^j;

5. Experimental Result

In this section, we compare the performance of VMs and containers for the concurrency condition that is the most significant characteristic of fog computing. The experiences verify that containers, which are the foundation of our architecture, are more suitable for fog computing than VMs. We run multiple Linpack programs to test the performance loss of multi-program parallel runtime to simulate temporary services. Next, we simulate long-term services with MySQL and use SysBench as the TD to access MySQL.

5.1. Experimental Configuration

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 a fog node, the operating system of the server is ubuntu16.04. In addition, we use Docker 1.12.6, QEMU 2.5.0 and Libvirt 1.3.1 for our experiments.

5.2. Results

First, we used Linpack [25] to test the performance of containers and VMs under the concurrency condition. In this experiment, GNOME is used to start multiple terminals simultaneously in order to simulate concurrency. We set up all containers/VMs to share all physical resources. For example, the resource of a container will be 16 GB memory and 8 cores CPU when there is only one container or VMs. If there are two containers or VMs, each container or VM will have, for instance, 4 GB memory and 4 cores CPU. Containers use Ubuntu 16.04 as the base image and the operating system of VMs is also Ubuntu 16.04. The array size of Linpack is 200×200. Each level repeats for ten experiments to calculate the average value of the result to avoid result occasionality. As we expected,

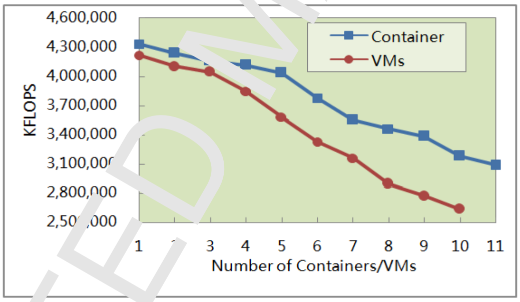
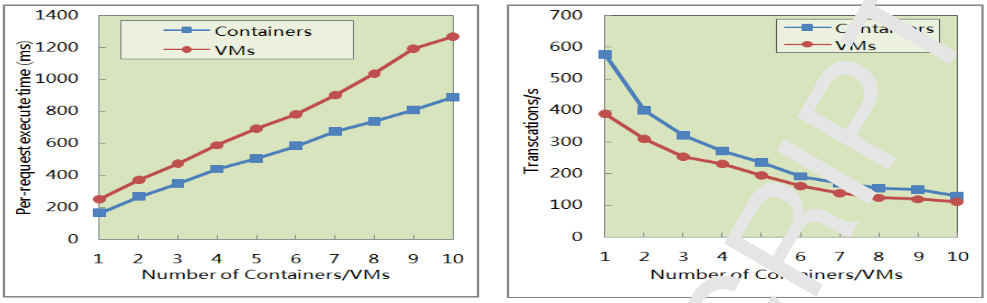


Figure 5: KFLOPS of containers/VMs

Fig. 5 shows that the performance of containers/VMs decreases with an increase in the number. Containers and VMs decrease with an equal rate from one to three, while from four to ten, the decrease in the containers is slower than the VMs, which proves that in the case of high concurrency, the CPU performance of a container is better than VMs'. Due to the performance limitations of the server, when we create 11 VMs, some VM failures render the complete result of 11 VMs unable to obtain.

Lastly, we simulate long-term services. In the long-term experimental environment, the resource allocation is the same as the configuration of the Linpack test. We employ



(a) Per-request execute time of Containers/VMs (b) Transactions of Containers/VMs

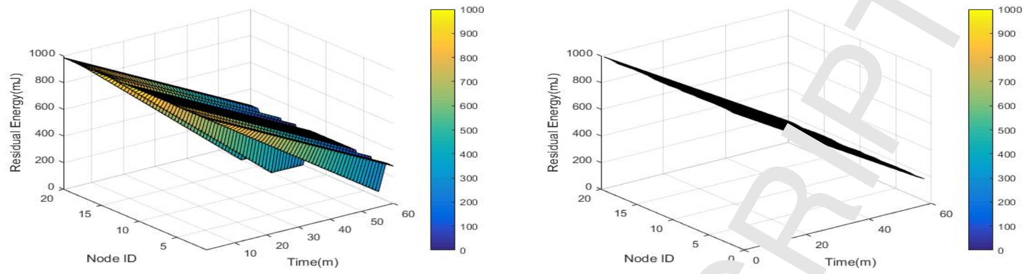
Figure 6: Per-request execute time and Transactions of Containers/VMs

MySQL as the long-term service application. We set up one container/VM to host one MySQL service. Note that we are using MySQL images directly instead of Ubuntu image as the base image. The database engine is configured to use InnoDB. Before the test, we use SysBench to insert 1,000,000 items in each database for a pressure test. Each MySQL has been tested by 100 threads to send requests concurrently and address a total of 100,000 requests.

Because of the average allocation of resources, the same level of the container or VM results are very close. Fig. 6(a) and Fig. 6(b) adapt the averaged value for comparison. We also did not test 11 containers/VMs, since some VM failures occurred when we are creating 11 virtual machines. Fig. 6(a) shows the per-request execution time as a function of the number of containers/VMs simulated by the OLTP of SysBench. The average execution time for each request increases with the number of containers/VMs increasing. However, containers perform is better than VMs from 1 to 10; the increase rate is slower than VMs. Fig. 6(b) plots the average number of transactions per second for all containers/VMs on each level. Initially, containers are considerably better than VMs, and VMs later gradually narrow the gap with containers.

5.3. Task Scheduling and Resource Management

In our simulation, there are 20 TDs to connect with a fog node, and each TD sends 50,100, 150, 200, 250 requests to the fog node in different experiments respectively. The configuration of the fog node is set with a schedule cycle of 100 ms. There are three types of data requests, 10, 20, and 30KB, respectively. In addition, $h_k = -40 \cdot (1/50)^4$, where -40 represents the path-loss constant and 1 represents the distance between TDs and the



(a) Maximum transmission power

(b) Power Balancing

Figure 7: Residual Energy of TDs

fog node, 50 is maximum transmission distance of TDs. The bandwidth w of each TD is 1 MHz, $o = 10^{-13}$ W, $\theta = 5$. The compared task scheduling algorithms are as follows:

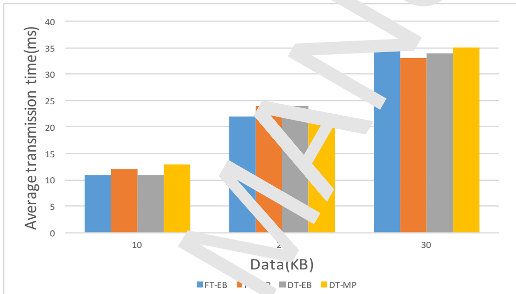
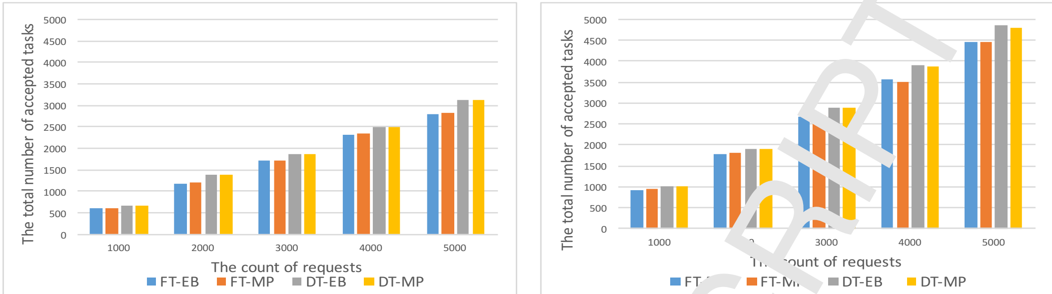


Figure 8: Average transmission time

1. Fixed threshold + Energy balancing(FT-EB): FT-EB uses fixed thresholds which is the most direct and simplest way for online task scheduling and energy balancing proposed in this paper.
2. Fixed threshold + Maximum power(FT-MP): FT-MP uses fixed thresholds for task scheduling and maximizes transmitting power for data uploading.
3. Dynamic threshold + Energy balancing(DT-EB): DT-EB uses dynamic threshold for task scheduling, which proposed by this paper and energy balancing for power management. This approach considers the high concurrency of fog nodes as well as the average transmission power of TDs.
4. Dynamic threshold + Maximum power(DT-MP): DT-MP uses dynamic threshold and maximized transmission power. This approach considers the high concurrency of tasks without energy balancing.



(a) 10 MIPS

(b) 20 MIPS

Figure 9: The total number of accepted tasks

Fig. 7 and Fig. 8 show the residual energy and the transmission time of TDs. Through the energy-balanced scheduling strategy proposed in this paper, the residual energy of each TD is evenly distributed. For the maximum transmission energy strategy, the energy of some TDs is exhausted prematurely. When other TDs have a large amount of energy, these TDs can only enter the dormant state and wait for their charge. And for the transmission time, they only differ by a few milliseconds and do not have a great impact on the entire data transmission.

Fig. 9(a) and Fig. 9(b) plot the number of task concurrency by 10 MIPS and 20 MIPS respectively. The results suggest that DT-EB and DT-MP, 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 approaches 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, leading 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.

6. Conclusion

This paper proposes multi-cloud to multi-fog architecture based on container and its corresponding implementation. It aims to address the challenge of private cloud developing. The key point of this architecture is using container as the resource unit to reduce the response time of requests and designing the cloud agent component to solve the problem caused by multiple tenants. In addition, a task scheduling algorithm based on energy balancing has been proposed to advance the life of WSNs without increasing the delay of tasks. We evaluate the performance of virtual machines and containers under the high concurrency, and the result shows that containers is better than virtual machines. Furthermore, we simulate the energy cost and task scheduling in the TDs and fog node respectively, the algorithm proposed in this paper can effectively balance the energy of TDs in the network.

Acknowledgement

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).

References

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

[2] F. Bonomi, R. Milito, J. Zhu, et al., Fog computing and its role in the internet of things., Edition of the Mcc Workshop on Mobile Cloud Computing ACM (2012) 13–16.

[3] L. M. Vaquero, L. Rodero-Merino, Finding your way in the fog: Towards a comprehensive definition of fog computing., Acm Sigcomm Computer Communication Review 44 (5) (2014) 27–32.

[4] B. I. Ismail, D. Jagadisan, M. F. Khalid, Determining overhead, variance & isolation metrics in virtualization for iaas cloud., Data Driven e-Science (2011) 315–330.

[5] M. Peng, S. Yan, K. Zhang, et al., Fog-computing-based radio access networks: Issues and challenges., IEEE Network 30 (1) (2016) 46–53.

[6] K. Liang, L. Zhao, X. Chu, et al., An integrated architecture for software defined and virtualized radio access networks with fog computing, IEEE Network 31 (1) (2015) 80–87.

[7] X. Hou, Y. Li, M. Chen, et al., Vehicular fog computing: A viewpoint of vehicles as the infrastructures., IEEE Transcations on Vehicular Technology 65 (6) (2016) 3860–3873.

[8] Z. Li, B. Chang, S. Wang, et al., Dynamic compressive wideband spectrum sensing based on channel energy reconstruction in cognitive internet of things., IEEE Transactions on Industrial Informatics (2018) 1–1.

[9] J. Nie, J. Luo, L. Yin, Energy-aware multidimensional resource allocation algorithm in cloud data center., KSII Transactions on Internet and Information systems 11 (9) (2017) 4165–4187.

[10] K. Xie, J. Cao, X. Wang, et al., Optimal resource allocation for reliable and energy efficient cooperative communications., IEEE Transactions on Wireless Communications 12 (10) (2013) 4994–5007.

[11] B. Yang, Z. Li, S. Chen, et al., Stackelberg game approach for energyaware resource allocation in data center., IEEE Transactions on Parallel and Distributed Computing 27 (12) (2016) 3646–3658.

[12] X. Sun, N. Ansari, Edgeiot: Mobile edge computing for the internet of things., IEEE Communications Magazine 54 (12) (2016) 22–29.

[13] L. Hou, S. Zhao, X. Xiong, et al., Internet of things cloud: Architecture and implementation, IEEE Communications Magazine 54 (12) (2016) 32–39.

[14] 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 16 (9) (2017) 5994–6009.

[15] Y. Mao, J. Zhang, K. B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE Journal on Selected Areas in Communications 34 (12).

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

[17] W. Felter, A. Ferreira, R. Rajamony, et al., An updated performance comparison of virtual machines and linux containers., 2015 IEEE International Symposium on Performance Analysis of Systems and Software (2015) 171–172.

[18] F. Ramalho, A. Neto, Virtualization at the network edge: A performance comparison., IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (2016) 1–6.

[19] S. Nadgowda, S. Suneja, N. Bila, et al., Voyager: Complete container state migration., IEEE 37th International Conference on Distributed Computing Systems (2017) 2137–2142.

[20] S. Osman, D. Subhraveti, G. Su, others., The design and implementation of zap: a system for migrating computing environments., ACM SIGOPS Operating Systems Review 36 (6) (2002) 361–376.

[21] Y. Luo, B. Zhang, X. Wang, et al., Live and incremental whole-system migration of virtual machines using block-bitmap., IEEE International Conference on Cluster Computing (2008) 99–106.

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

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

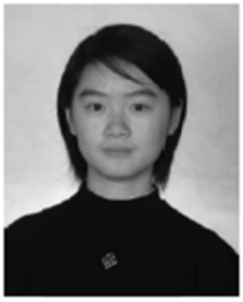
[24] A. Miettinen, J. Nurminen, Energy efficiency of mobile clients in cloud computing, USENIX Association Berkeley, HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, Boston, 2010, pp. 4–4.

[25] Linpack, `https://github.com/ereyes01/linpack`.

Biography of the Authors

JUAN LUO (M'10) received the Bachelor's degree in electronic engineering from the National University of Defense Technology, Changsha, China, in 1997, and the Master's and Ph.D. degrees in communication and information systems from Wuhan University, Wuhan, China, in 2000 and 2005, respectively. Currently, she is a Professor and Doctoral Supervisor with the College of Computer Science and Electronic Engineering, Hunan University, Changsha. She is also the Director with the Department of Network and Information Security, Hunan University. From 2008 to 2009, she was a Visiting Scholar with the University of California at Irvine, Irvine, CA, USA. She has authored/coauthored more than 40 papers. Her research interests include wireless networks, cloud computing, and wireless sensor networks. Dr. Luo is a Member of the Association for Computing Machinery (ACM) and a Senior Member of China Computer Federation (CCF).

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, Hunan, China. He is currently receiving for a PhD degree in college of information science and engineering, Hunan University, Hunan, China. His research is focused on cloud computing and wireless virtual network.



JIUYU HU received the B.E. degree in communication engineering from Hunan University, Changsha, China, in 2009, and the M.E. degree in computer applications technology from Chongqing Jiaotong University, Chongqing, China, in 2013. Currently, she is pursuing the Ph.D. degree at the College of Computer Science and Electronic Engineering, Hunan University. Her research interests include wireless networks.

CHUN WANG received the B.E. degree in China University of Mining and Technology in 2013 and 2017from College of computer science and technology, Xuzhou, China. She is currently receiving for a PhD degree in college of information science and engineering, Hunan University, Changsha, China. Her research is focused on wireless networks.

XIN FAN received a bachelor's degree in science and technology from Anhui Jianzhu University in 2017. He is currently acquiring a master's degree from the Hunan University for information science and engineering. His research focuses on cloud computing and resource scheduling in wireless data centers.

XUAN LIU received the B.E. degree and M.E. degree in electronic engineering from the National University of Defense Technology, Changsha, China, and the Ph.D. degrees in Hong Kong Polytech University, respectively. Currently, she is a Professor with the College of Computer Science and Electronic Engineering, Hunan University, Changsha. Her research interests include wireless networks, cloud computing, and wireless sensor networks. Dr. Luo is a Member of the Association for Computing Machinery (ACM) and a Senior Member of China Computer Federation (CCF).

HAIBO LUO received the B.E. degree in Communication Engineering from Wuhan University of Technology, China, in 2006, the M.E. degree in Information and Communication Engineering from Hunan University, China, in 2009. He is currently studying for a Ph.D.

degree in College of Physics and Information Engineering, Fuzhou University, Fuzhou, China. His research interests include Internet of Things, Edge computing and Mobile computing.