J Grid Computing (2019) 17:699–726
https://doi.org/10.1007/s10723-019-09499-7

Scheduling Algorithms for Heterogeneous Cloud Environment: Main Resource Load Balancing Algorithm and Time Balancing Algorithm

Weiwei Lin · Gaofeng Peng · Xinran Bian · Siyao Xu · Victor Chang · Yin Li

Received: 27 October 2018 / Accepted: 29 October 2019 / Published online: 19 November 2019

Abstract Cloud computing and Internet of Things (IoT) are two of the most important technologies that have significantly changed human's life. However, with the growing prevalence of Cloud-IoT paradigm, the load imbalance and higher SLA lead to more resource wastage and energy consumption. Although there are many researches that study Cloud-IoT from the perspective of offloading side, few of them have focused on how the offloaded workload are dealt with in Cloud. This paper proposes two IoT-aware multi-resource task scheduling algorithms for heterogeneous cloud environment namely main resource load balancing and time balancing. The algorithms aim to obtain better result of load balance, Service-Level Agreement (SLA) and IoT task response time and meanwhile to reduce the energy consumption as much as possible. They both are devised to assign single task to a properly selected Virtual Machine (VM) each time. The task placed in a pre-processed queue is assigned sequentially each time. And the VM selection rule is carried out based on the newly inventive ideas called relative load or relative time cost. Besides, two customized parameters that influence the result of pre-process tasks are provided for users or administrators to flexibly control the behavior of the algorithms. According to the experiments, the main resource load balancing performs well in terms of SLA and load balance, while time balancing is good at saving time and energy. Besides, both of them perform well in IoT task response time.

Keywords Heterogeneous cloud · Task scheduling · Load balance · Multi-resource scheduling · SLA · Energy consumption · IoT

W. Lin (✉) · G. Peng · S. Xu
School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
e-mail: linww@scut.edu.cn

W. Lin
Guangdong Luan Indusrty and Commerce Co.,Ltd., Guangzhou, China

X. Bian
Shanghai Jiao Tong University, Shanghai, China

Y. Li
Institute of Software Application Technology, Guangzhou & Chinese Academy of Sciences, Guangzhou, China

V. Chang
School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough, UK

1 Introduction

As a popular way of computing paradigm and providing IT services, cloud computing [1–3] can provide users with on-demand and easy access and also stand out in cost savings. With the growing prevalence of the application of the cloud computing upon people's daily life, the requirements for cloud computing are also becoming higher. Researchers who study cloud computing, therefore, aim the goal of making cloud computing capable of better performance while consuming less energy. To achieve this goal, one of the necessary steps that needs

to be done is that the assignment policy of tasks should be reasonable. On the one hand, reasonable tasks scheduling policy can raise resource utilization which can avoid less idle resource. On the other hand, it can shorten tasks completion time and reduce energy consumption. Unlike cloud environment, devices across heterogeneous network domain are loosely connected in IoT environment. In such environment, efficient routing protocol can be critical for saving energy [4]. Baker et al. [5] first proposed a novel network-based routing algorithm which can find optimally energy-efficient route to transfer big data to cloud for further processing, namely GreeDi. Based on GreeDi, Baker et al. [6] further designed a novel reactive routing protocol for vehicle communications. It is capable of selecting the most efficient routes for reducing power consumption. The rapid growth of cloud computing and IoT greatly affects people's daily life. The technology of IoT connects things together, even including human beings. Additionally, it improves the intelligent management of available resources [7]. Therefore, the integration of cloud and IoT is also necessary. Some researchers present the need for the integration and demonstrate the possible future directions. Botta et al. [8] proposed a paradigm called CloudIoT, which could integrate cloud with IoT. The new paradigm renders IoT benefit from unlimited resources of cloud to break through its native constrains. In this paradigm, offloading is one of the key techniques that links IoT to cloud.

1.1 The Role Offloading Plays in Cloud-IoT

Large-scale deployment of IoT devices faces the issues caused by energy and performance constraints. Fortunately, offloading is a promising technique that can effectively address these issues. Zhao et al. [9] focused on computational offloading for mobile devices. The authors argued that the need for mobile devices to run applications that require high computational load is growing rapidly. Mobile phones still have limited computing resources compared to traditional computing devices, such as desktop computers. The limitation is primarily due to their physical size. Mobile devices can offload tasks to the cloud, yet it may degrade users' experience if the delay is intolerable. To alleviate this problem, fog computing has emerged and become a good choice under some certain circumstances. However, compared to the cloud, the fog still has less resources. It might further bring negative impacts on the amount of energy consumed by fog nodes. As is known to us, the less powerful a device is, the longer time it will take to execute the same workload. Therefore, the authors proposed an offloading scheme that evaluates the energy consumption generated by the fog and the cloud before offloading. After that, the destination is finally decided.

Hasan et al. [10] presented a bunch of incentive-based offloading schemes for mobile nodes. The authors stated that the paradigm in communication and computing has changed because of the proliferation of IoT devices. An ad hoc cloud of IoT and other computing devices called Aura was proposed in the article. It can exploit the idle resource, such as computing resources from underloaded devices, as much as possible. The core of the authors' idea lies in how to make such devices accept offloaded tasks from other devices so that the idle resources can be outsourced. Therefore, a mechanism of rewarding in the form of crypto-currency was employed in the article. Runtime computational offloading is a complicated process. It can bring side effects because of the unavoidable cost of offloading which the articles mentioned above did not consider. To resolve this issue, M. Shiraz et al. [11] proposed a computational offloading framework for mobile cloud computing that considers component migration overhead. It leverages application processing services in the cloud data center to migrate the least number of application instances at runtime.

Besides the Aura mentioned above, there are many other computation offloading frameworks surveyed in [12], such as Avatar, MALMOS, and MoSeC etc.

1.2 Offloaded Workload as a Type of Workload in Cloud

It can be observed that there are many researches that focus on offloading, yet most of them focus on the IoT side from which the offloading starts. In this paper, we will focus on the cloud side.

Once the offloaded computation workloads arrive in cloud, they can be seen as common as other traditional tasks or jobs. As we all know, the behaviors of workloads in clouds are always variant [13]. Among them, some belong to the workload offloaded from IoT device. The reason for uploading workloads from IoT devices is that some workloads require too much computing resources. Consequently, the device may choose to upload several workloads to the Fog or Cloud. In many ways,

they can be deemed as common workloads. Thus, it is necessary to consider how workloads are dealt with on the data-center side.

1.3 Researches on Load Balancing Using VM Migration Approaches

When talking about cloud, we usually consider load balancing as an important technique used to distribute workload among VMs. Without it, the unbalanced loads in datacenter might cause resource wastage, performance degradation and SLA conflicts [14]. As a result, the usage of the technique can further improve the utilization of servers and better guarantee Quality of Service (QoS). To address the performance degradation problem of cloud servers, there are a lot of discussions about load balancing. Some of the researchers in this field focus on virtual machine allocation or virtual machine migration to achieve load balancing. This area will be reviewed below.

In [15], the authors first stated the well-known reason that results in unbalanced loads. They emphasized that it is the disproportionate utilization of resource that leads to load imbalance. Correspondingly, the idea of skewness was introduced to indicate the magnitude of asymmetry in resource usage of a host. They formulated the problem as bin packing problem. By decreasing the value of skewness, the overall utilization of servers became better.

In [16], a self-adaptive virtual machine consolidation scheme built on multi-threshold adjustment mechanism was proposed. The mechanism was realized by comparing the predicted requests for each multi-dimensional infrastructure resource with their current resource usage status. The dynamic multi-thresholds were used for physical machine selection, which is the first step in VM integration. Besides, a modified virtual machine selection and allocation algorithm were proposed. Each of the algorithms belongs to the next two steps of VM consolidation.

In [17], the authors proposed an advanced version of the modified best fit decreasing (MBFD) algorithm [18], which focuses on reducing the number of active servers and leaves a stable host to every VM. In this way, the frequency of unnecessary migration can be reduced. Prior works which used the bio-inspired methods, such as Firefly optimization [19], also contribute a lot to this field. In [19], the authors employed the meta-heuristic technique to improve the energy efficiency for live migration of VMs. It was aimed to minimize the number of VM migrations and the number of hosts. Unfortunately, their experiments did not evaluate some inborn drawbacks of this type of technique, such as the convergence rate and the tendency to be locally optimized.

1.4 Researches on Load Balancing Using Task Scheduling Approaches

While server consolidation achieves considerable effectiveness in capping energy consumption and operating cost, it can also cause performance degradation if it is not devised properly [20, 21]. On the one hand, server consolidation attempts to consolidate virtual machines on some servers, making it more likely for machine overload to occur. On the other hand, the shutdown time and communication cost incurred by server consolidation is unavoidable and needs to be handled properly to satisfy the delivered QoS [22, 23].

Besides VM, there are also research focusing on task allocation to achieve load balancing. In [24], the authors treated the problem of assigning tasks to virtual machines as a matrix. After putting it like that, a problem called unbalanced assignment need to be considered. Hence, the authors decided to employ the Hungarian method to solve the problem. Some simple but well-recognized heuristics, such as Round Robin, Random Selection and First Come First Serve, have been applied in this field. Malik et al. [25] developed a load balancing algorithm based on Round Robin. Faced with the drawback of Round Robin, the tasks are first prioritized based on required resources or processors, number of users, runtime, job type, user type, and software used cost. Then, they are assigned to various available hosts in Round Robin fashion. While Mittal et al. [26] incorporated the advantages of existing algorithms such as Max-Min, Min-Min and RASA. The authors proposed a scheme to carry out load balancing amongst the various types of resources to achieve better make-span by integrating the advantages of these existing algorithms.

To better utilize the underlying information about the tasks and VMs, Adhikari et al. [27] developed a new heuristic-based algorithm for load balancing. The algorithm is split into two phases: server configuration and task-virtual machine mapping. The first phase is designed for better utilizing the resources. This strategy helps to make efficient use of resources. In the second phase, a queueing model is employed to minimize the task response time and the total make-span.

In order to solve the problem of load imbalance via exploiting prior knowledge, Alaei et al. [28] proposed and designed a reactive and active scheduling framework that relies only on prior knowledge related to task scheduling for load balancing. But the prior knowledge is usually not rich. On the other hand, Singh et al. [29] concluded that to better utilize the resource in Cloud, their types (homogeneous and heterogeneous) should be considered. Motivated by this consideration, Panda et al. [30] proposed SLA-MCT and SLA-Min heuristic algorithms for load balancing scheduling. The algorithms are devised for heterogeneous environment. Both techniques use the execution time and cost as SLA parameters.

In addition, some other researches might employ meta-heuristic (EA and other bio-inspired methods especially) to transform the bin packing or bin-packing-like problem into near optimal solution space searching. For example, Zhou et al. [31] proposed a multi-objective hybrid BCO algorithm that solves the service combination and optimal selection problem. It takes QoS and energy consumption into account. In the utilized bee phase, an enhanced solution update equation with different dimensions of perturbation is used. The method has its mechanism to avoid being trapped in local optima and the QoS is guaranteed. Aiming to reach optimum task scheduling result, Shojafar et al. implemented a hybrid approach called FUGE [32]. The approach can perform optimal load balancing considering execution time and cost. The core of the approach lies in its exploitation of fuzzy theory to modify the standard genetic algorithm. FUGE considers VM processing speed, VM memory, VM bandwidth, and the job lengths when scheduling tasks. Despite all the advantages of the above methods, it leads to low load balancing. In order to achieve better load balancing, some researches employ the Inverted ACO [33]. The scheme proposed by Asghari et al. [33] showed low cost and high load balancing. However, it also shows high execution time and does not consider QoS.

1.5 The Motivation of Using Heuristic-Based Approach

In spite of the many good qualities of some meta-heuristic algorithms applied in some situation, which in most cases tend to be suitable for global optimization, they are not always successful or efficient. Moreover, if a proposed method is designed to have less scheduling overhead and to be as less complex as possible, it won't help. That is because they tend to use many equations and complex operators [34]. Also, the tuning of parameters also increases the complexity. In this paper, the proposed method, which is based on heuristic theory, is employed to process the assignment of a batch of tasks to virtual machines. We aim to develop a VM heterogeneity-aware method that concerns load balancing and multi-resource task allocation. And most importantly, it should also be IoT workload-aware. Hereby, we only discuss and compare the heuristic methods.

1.6 Problem Statement

In the practice of Cloud-IoT, energy consumption and QoS can be improved from the perspective of task scheduling in data-center. The new paradigm consists of two processes of workload being sent from end devices and workload being processed by the more centralized tiers. The existing approaches mentioned above present how the energy consumption can be reduced and how the QoS can be better through making the IoT end devices submit or offload workload to the data-centers or other edge servers. This field is well researched. The problem is that they proposed many schemes to offload workload from IoT devices but the issues of how the workload should be tackled after submitting are open.

The issues that we find include the following points:

- Bio-inspired meta-heuristic schemes are popular. However, they can be too complex for problem solving and be unstable in results.
- Users prefer having more powerful and more flexible control of the scheduling results.
- The simulation tool for verifying multi-resource oriented schemes is few.
- There is no proper mathematical formulation to model the task and VM in the heterogeneous cloud.
- Processing IoT offload workload from the perspective of task scheduling in data center means that our scheme should be IoT-aware. The concern is how to improve the IoT task response time without bringing too much negative impact to others when IoT tasks and other common tasks are mixed together for scheduling.
- It is really difficult to make the scheme produce the best results of SLA, QoS, energy consumption, and load balancing at the same time. But we can choose

to make some trade-offs among them to some extent.

1.7 Aims and Contributions

In summary, the algorithms proposed in this paper are aimed to be heuristic-based, IoT-aware and focusing on multi-resource task scheduling for heterogeneous environment. The main novelty of our algorithms lies in the way of selecting the proper task-VM mappings from one queue of pre-processed tasks and the other set of VMs. The pre-processing includes two phases in which the first step is ordering the tasks by priority and the second step is re-ordering the tasks by the category of task. The priority of task reflects the urgency degree of task to be assigned and the category includes IoT task and Common task. There are two parameters provided in these two steps respectively and they render the users flexibly control the pre-processed results. The selecting strategy is according to the task sequential positions in the pre-processed queue and the mapping strategy is a modified version of greedy algorithm. All the details will be mentioned in the following sections.

In the meantime, the results, such as SLA, energy consumption, and make-span, should be compared with other existing algorithms to verify its effectiveness. Moreover, the users or administrators should be able to flexibly adjust the behaviors of the algorithms. To help us take the first step, the comparisons of recent years' related researches should be performed. Only in this way, we can find out the clues and the related works that can enlighten us. Table 1 provides an overview picture of the existing studies.

An architecture that connects IoT devices, Fog, and Cloud is provided. As shown in Fig. 1, there are four major components on the cloud, namely people that manage the platform, a batch of tasks, VMs on hosts, and the task scheduler. At the very beginning, each task is stored into the task queue. After that, the queue will be sorted and reordered based on the given α and k. The task scheduler is actually the process of mapping tasks and VMs. They are the core ideas of this article's proposed algorithms. Both will be detailed in the following sections. There are also some middlewares in the figure, such as smart gateway, assisting the cloud in forwarding the offloaded workload to fog [35]. However, we only discuss the cloud here. The only reason they are included in the figure is to keep the universal view related to IoT. It should be noted that we reasonably assume that all the workloads submitted to the data-centers are labelled. As a matter of fact, it is not a difficult technique to label workloads as end devices or end users can specify them right before submitting.

The specific contributions of this paper are listed as follows:

- We design two algorithms for multi-resource task scheduling in heterogeneous clouds: the main resource load balancing and the time balancing. According to the experiments, compared with the existing approaches, the former algorithm produces lower SLA and better trade-off between energy consumption and make-span. Since they are IoT-aware, both can produce low response time for IoT tasks.
- To realize the method introduced by the above research results, we propose several mathematical models to facilitate the simulation of heterogeneity-aware load balancing. The models consider multi-resource and are heterogeneity-aware, which means they consider hosts' or VMs' different capacity of resources.

We provide two parameters, namely α and k, to enable users or administrators to flexibly control the behaviors of the two algorithms. Firstly, the parameter α determines the importance between the task run time and the CPU demands. Secondly, the parameter k determines the importance of IoT tasks in execution queue. Users or administrators can tune them according to the variation of the recently submitted workload. As can be seen, the algorithms are IoT-aware.

This paper is organized as follows. In section 2, the related works on task scheduling and the tool used for developing are discussed. In the third section, two proposed algorithms are detailed. And in section 4, the core concepts of the theory are further detailed and analyzed. Evaluations of our study are presented in section 5 and this paper is concluded in section 6.

2 Related Works

2.1 Energy Consumption Optimization in Cloud Computing

Based on the limitations of previous researches on energy-performance trade-offs in the elastic cloud

Table 1 Comparison of some existing studies

	Category	Energy-aware	IoT-aware	Load-balancing	Heterogeneity-aware	Multi-resource-aware	Make-span-aware	Response time-aware	Scheduling overhead-aware	SLA-aware	Weaknesses	Contributions
14	Dynamic and VM allocation	Yes	No	Yes	Yes	Yes	No	No	No	Yes	CPU energy is considered only.	New concept named skewness measuring the eveness of server resource utilizat-ion.
15	Dynamic and VM scheduling	Yes	No	No	Yes	Yes	No	Yes	Yes	Yes	Unstable LR-based predic-tion for certain workloads	Weighted VM migration and dynamic multi-thresholds
16	Static and VM sched-uling	Yes	No	No	No	No	No	No	Yes	Yes	Reduce VM migration tim-es by imposing a static thr-eshold.	Reduce the num-ber of overall migrations.
28	Dynamic and task scheduling	No	No	Yes	No	No	Yes	Yes	No	No	Too simple combinations of algorithms.	Switching the famous algorith-ms to deal with the changing sit-uations.
29	Dynamic and task scheduling	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Server knowledge is needed.	Minimizee the make-span by scheduling the waiting tasks efficiently.
22	Dynamic and multi--objective	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	Unstable stra-tegy based on solution space searching.	Develop a par-ameter adaptive strategy.
23	Dynamic and task scheduling	No	No	Yes	Yes	Yes	Yes	Yes	No	No	Unstable strategy based on solution space searching	Devise a fuzzy-based steady-state GA.
36	Dynamic and VM scheduling	No	No	Yes	Yes	No	No	No	Yes	Yes	Peak similarity only based on CPU.	Place VM based on peak workload similarity.
37	Static and task scheduling	Yes	No	Yes	No	Yes	Yes	No	No	No	Heterogeneity is not consi-dered.	Develop a simulation tool for multi-resource tasks and a task balance strategy.
40	Static and task scheduling	Yes	No	No	Yes	Yes	Yes	Yes	No	Yes	Support of elastic resources is needed.	Compare constraint and objective vectors in different dimensions via normalization.
42	And multi--objective	Yes	No	No	Yes	No	Yes	Yes	No	No	Might lead to load imbalance and lots of knowledge is needed.	Energy-performance trade-off
43	Dynamic and task scheduling	No	No	Yes	Yes	No	Yes	Yes	No	No	Load imbalance.	Increase resource utilization, and achieve higher throughput

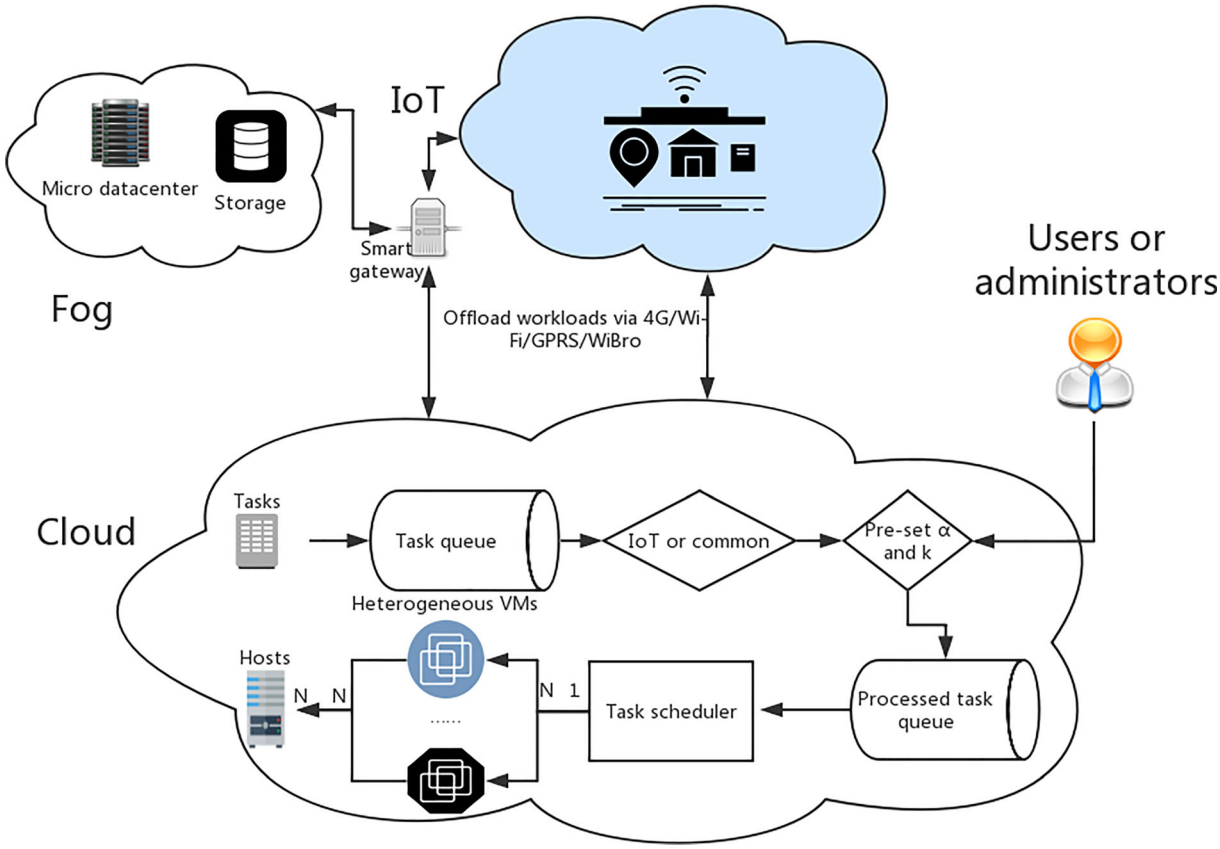

Fig. 1 Overview of the proposed algorithms

environment, Yang J et al. [36] proposed an energy-performance trade-off task scheduling algorithm. Firstly, through virtualization technology, they successfully simulated energy-aware task scheduling problem in clouds. Then, an energy-aware resource provisioning algorithm was developed for generating real-time tasks' available allocation plans by incorporating the elasticity of virtual resources. Finally, an energy-performance trade-off task scheduling algorithm for final solution selection based on multi-object optimization was developed. In the task scheduling process, the idea of Max-Min fairness was employed, yet not the simplest version of Max-Min. It should also be noted that their methods are IoT-agnostic and their resource provisioning method employs the VM migration technique which is not used in our proposed schemes.

To be more energy efficient, Congyang Chen et al. showed their detailed researches on the techniques and scheduling algorithms of workflow optimization in [37]. They summarized the technique's theory and applied field, and divided them into three categories in terms of single-objective optimization, multi-objective optimization and heuristic based. F. Juarez et al. [38] assumed that the resource provider or user can pre-specify a factor deciding the energy-performance importance. They proposed a scheduler whose goal is to minimize a bi-objective function. The function combines the energy consumption and the make-span. They are combined by the importance factor. Hussain et al. [39] proposed a task batch scheduling algorithm in dynamic way. By considering the computing share of virtual machines, the tasks are assigned to VMs based on the idea of Max-Min fairness. However, they all are either IoT-agnostic or not multi-resource scheduling.

2.2 MultiRECloudSim

Among those many cloud simulators, CloudSim [40] is one of the most widely used platforms. As a matter of fact, the platform supports various types of resources including CPU, RAM, bandwidth, etc. Numerous previous studies are realized by CloudSim, such as research

in [41]. The article proposed a VM placement scheme which can place VMs with similar workload peaks together and the authors verified the effectiveness of the scheme with CloudSim. However, it should be noted that CloudSim won't regard it as mistakes even if the required RAM or bandwidth exceeds what VMs or even hosts actually have. In addition, CloudSim-based energy simulation only supports CPU-aware scheduling and a single task running in the simulation which assumes that there is just one task performed by a virtual machine. Hence, it limits the CloudSim-based energy simulation. In order to resolve the underlying problem of multi-resource-agnostic mentioned above, W. Lin et al. proposed a simulator named MultiRECloudSim which considers multi-resource tasks' scheduling and energy simulation.

The most related research developments in MultiRECloudSim are listed as follows [42]:

1) A new task model is designed for multi-resource. IO is added as a new type of resource in the model. The model regards that the fully-met demands of IO, bandwidth, and RAM are indispensable for a task to start while the CPU's demand is not. But the amount of CPU allocated to a task will influence the speed of execution. Those demands exist in SimCloudlet object respectively as attributes in terms of IO, BW (bandwidth), RAM, MIPS. Among them, only MIPS supports both static and dynamic workload. Furthermore, the current requested MIPS with the dynamic workload is calculated by the sum of the product of the requested MIPS utilization of a task and the standard MIPS of the task rather than the product of the total requested utilization of MIPS overall running tasks and the MIPS allocated to the VM. Those changes definitely make the simulation closer to the real-world scenario.
2) Two new definitions are introduced: the normal resource load of task (NRLT) and the normal resource load of VM (NRLV). The former one is a metric used to measure how much workload for the resource a task has while the latter one is a metric used to measure how much workload for the resource a VM has. NRLT is calculated by the product of the estimated execution time of task x and the normalization of the average demand of task x for resource c_i, as shown in Eq. (1). NRLV is calculated by the sum of the normal resource load of all running tasks and the tasks to be executed in the VM, as shown in Eq. (2). The meanings of notations are mentioned below. All the meanings of the notations are listed in Table 2.

$$Load(x, c_i) = time(x)Normal(x, c_i) \tag{1}$$

$$Load(vm_i, c_j) = \sum_{x \in E(i)} Load(x, c_j) \tag{2}$$

3) Based on CloudSim's previous task scheduler, a new task scheduler suitable for multi-resource allocation and multi-resource task scheduling. It supports the task waiting queue and uses the resource allocators to manage and allocate resources. The resource allocators are divided into three categories as CPU allocator, RAM allocator, bandwidth allocator, and IO allocator. Since the demand for CPU must be met, it means that other types of resource are reserved except CPU. When a task starts executing in reservation mode, the largest amount among CPU demands of all tasks will be allocated to it and afterwards, each task will be allocated the same amount of CPU in each time slot. The reservation scheme can fully guarantee QoS but result in resource idleness. Conversely, the non-reservation scheme is an on-demand scheme so it cannot guarantee QoS.
4) It supports multi-resource including CPU, RAM, IO, and bandwidth energy simulation. The basic workflow of carrying out the simulation is listed as follow: firstly, the energy consumption is calculated for every fixed time interval. According to the first resource utilization and the last resource utilization in the interval, the energy consumption is calculated by linear fitting at resource-power model.

There is an underlying problem of load balancing algorithm proposed in [42]: it doesn't consider the difference that the same task impacts on heterogeneous VM. For example, given VM A with 2000 MIPS and another VM B with 3000 MIPS, a task with CPU demand in term of 1000 MIPS causes different impacts on VM A and VM B. For VM A, its CPU utilization will rise from 0 to 50%, whereas VM B is 0 to 33.3%. Hence,

Table 2 Meanings of the notations

x	A task. Specifically, a cloudlet in MultiRECloudSim.
x^i	A task whose index is i in the executing task batch.
x^i_{IoT}	An IoT task whose index is i in the executing task batch.
S	The task execution queue or cloudlet-list in CloudSim.
c_i	Any type of resource. Such as c_{cpu}.
c_i^*	The normal value of corresponding resource c_i. Such as c^*_{cpu}.
vm_i	Any of the VMs.
p_j	Any of the hosts.
α	The parameter for tuning tasks' priorities.
target	Current task's main resource.
$c_i(x)$	The average demand of task x for resource c_i. Such as $c_{cpu}(x)$.
$c_i(p_j)$	The total amount of resource c_i in host p_j.
$c_i(vm_j)$	The total amount of resource c_i in VM j.
$time(x)$	The estimated execution time of task x.
$Normal(x, c_i)$ or $N_x^{c_i}$	The normalization of task x's average demand for resource c_i. Used for deciding tasks' main resource.
$Normal(x, c_i, vm_j)$ or $N_x^{c_i}(vm_j)$	The normalization of task x's average demand for resource c_i in VM j
$Load(x, c_i)$	The normal resource load of task (NRLT).
$Load(x, c_i, vm_j)$ or $Load(x, vm_i)$	The relative load of task x for resource c_i upon vm_j or the relative execution time of task x upon VM i
$Load(vm_i, c_j)$ or $Load(vm_i)$	The total relative resource load in a VM or the total relative execution time for VM i.
$E(i)$	The set containing all the running and waiting tasks in vm_i.
$length(x)$	Task x's length.
CloudletList	The task queue in CloudSim.
$Priority(x)$	Task x's priority.
REFERENCE _ TIME or *R _ T*	The given reference value of a task's running time.
REFERENCE _ CPU or *R _ C*	The given reference value of a task's MIPS.
$u\ (vm, c_i)$	The resource c_i utilization of a VM.
$W\ (vm, c_i)$	The load of a VM.
$u\ (cpu_i)$	Any of the PE's utilization.
$\Delta W(vm, c_i)$	The increased load of a VM.

the previous calculation method [42] on normal resource load of VM disregards the VM heterogeneity. As we all know, VMs might differ in configurations of CPU, RAM, IO, and bandwidth.

Accordingly, a task assigned to different VMs should present different loads for VMs. Fixed load value can't reflect non-uniform distribution of resources. In other words, load value should vary from VM to VM.

Besides, when assigning a batch of tasks to VMs, the tasks' make-span depends on the VM that finishes the last one task. If there were some tasks that finished executing very early while some very late, the idle resource of early finished VMs would be wasted.

Therefore, each VM's make-span should be closed to each other's. To achieve this, basically, we will arrange assignment of tasks with long execution time earlier and the shorter ones later. In this way, we can align VMs' make-spans. In addition to the length of tasks, it is extremely important to make full use of VM's performance at every moment. For example, a VM with CPU utilization always at 90% is compared with a VM with CPU utilization always at 60%. There's no doubt that the former one makes better use of resource. Of course, the former one can finish the execution of tasks earlier supposed that the two VMs have the same MIPS.

In the context of static batch task scheduling, the resource in which a task is assigned to are predetermined and remain unchanged till current batch completes. The existing load balancing algorithm tended to remain the same coping behaviors to varying workload till the end. But that is not what a cloud administrator wants to see.

Furthermore, the existing load balancing algorithm is not IoT-aware. The offloaded workload from IoT devices is also seen as the traditional or common workload in the eye of data-center. However, they are different from the common workload. The reason why some workload is offloaded from IoT device is that it requires too much computing resources. So, the device might choose to upload some workload to the Fog or the Cloud. In the meantime, the device should carefully consider the overhead caused by workload partitioning and workload uploading. Thus, the offloaded workload is usually more delay-sensitive than the common workload [43].

Based on the concerns mentioned above, we now propose main resource load balancing algorithm and time balancing algorithm for heterogeneous environment.

3 Main Resource Load Balancing Algorithm and Time Balancing Algorithm in Heterogeneous Environment

3.1 Design of Main Resource Load Balancing Algorithm in Heterogeneous Environment

3.1.1 The Introduction of Relative Load

In our theory, x means any one task, c_i means i_{th} type of resource, $c_i(x)$ means the average demand of task x for resource c_i, c_i^* means the normal value of resource c_i, $length(x)$ means task x's length, $c_{cpu}(x)$ means the average demand of task x for MIPS, p_j means j_{th} host, $c_i(p_j)$ means total amount of resource c_i in host p_j, $c_i(vm_j)$ means total amount of resource c_i in VM vm_j. All the tasks submitted to data-center are reasonably assumed already labelled. We use $Normal(x, c_i)$ to denote the normalization of average demand of task x for resource c_i, as shown in Eq. (3). Besides, we use $Normal(x, c_i, vm_j)$ to denote the normalization of average demand of task x for resource c_i in vm_j, as shown in Eq. (4).

$$Normal(x, c_i) = \frac{c_i(x)}{c_i^*} \tag{3}$$

$$Normal(x, c_i, vm_j) = \frac{c_i(x)}{c_i(vm_j)} \tag{4}$$

Accordingly, there is Eq. (5).

$$Load(x, c_i, vm_j) = time(x) \times Normal(x, c_i, vm_j) \tag{5}$$

Here, $Load(x, c_i, vm_j)$ denotes the relative load of task x for resource c_i upon vm_j. Then we easily deduce an equation, as shown in Eq. (6). To be more straightforward, an example is presented below. Assuming that there are two types of tasks (x_1 and x_2) whose estimated demand for CPU are 1000 and 500 and estimated length are 2000 and 1000, respectively, they are both ready to be scheduled to vm_1 and vm_2 whose capacity of CPU are 2000 and 3000, respectively. Based on Eq. (4), there are the distinct normalization values of x_1's demand for CPU in vm_1 and vm_2: 0.50 and 0.33. Then according to Eq. (5), there are the distinct relative CPU loads of x_1 in vm_1 and vm_2: $Load(x_1, cpu, vm_1)$ =1.00 and $Load(x_1, cpu, vm_2)$= 0.66. Considering the machine heterogeneity, the same type of task can have different impacts on different machines. Similarly, it is easy to get the equation: $Load(x_2, cpu, vm_1)$=0.50. By Adding this value to $Load(x_1, cpu, vm_1)$, the total relative CPU load of is got, which is reflected in Eq. (6).

$$Load(vm_i, c_j) = \sum_{x \in E(i)} Load(x, c_j, vm_i) \tag{6}$$

$E(i)$ denotes a set containing all running and waiting tasks in vm_i.

3.1.2 The Introduction of Parameter α

$Priority(x)$, as shown in Eq. (7), denotes task x's priority which indicates the greater the priority is the earlier the task x will be assigned to a vm.

$$Priority(x) = \frac{\alpha \times time(x)}{R\_T} + (1-\alpha) \times \frac{c_{cpu}(x)}{R\_C} \tag{7}$$

In Eq. (7), α denotes a value which is between [0,1], R_T denotes $REFERENCE\_TIME$, and R_C denotes

REFERENCE _ CPU. α is employed to balance the relative importance between task execution time and requested CPU. It is not difficult to find out that when α=0, the priority of a task is only determined by CPU. And the priority of a task is only determined by estimated execution time when α=1. It should be noted that CPU consumes the most energy of a server. In other words, the tuning of α can be deemed as balancing the algorithm's inclination to time-saving or energy-saving.

time(x) denotes task x's estimated execution time and *REFERENCE _ TIME* denotes the given reference value of a task's run time *REFERENCE _ CPU* denotes the given reference value of a task's MIPS. Both reference values are selected empirically and then used to calculate the priority. The reference values are selected by the users empirically. For example, the reference value of CPU may take the initial value of the resource allocated to VM.

3.1.3 The Introduction of Parameter k

The parameter α is employed to determine all the tasks' execution priorities. But unlike α, the given parameter k is employed to determine the IoT tasks' execution priorities. It is only used after the execution queue has been sorted by Eq. (7). In fact, it is a reordering process that adjusts the already sorted tasks' place in the queue. As can be seen from the constraint (8), k should be greater than the ratio by the sum of all the IoT tasks' index in the queue and the sum of all the tasks' index in the queue. To explain this constraint more clearly, an example is provided. Assuming that k is 0.4 and there are 3 sorted tasks: a, b, and c. c is an IoT task while others are all common tasks. Based on the right side of the inequality, the ratio is 0.5. It is obvious that the constraint is not satisfied and the reordering is needed. Based on our reordering scheme, c is placed 1 position leftward and the final task list becomes acb. Similarly, the final list becomes cab if the k is 0.2 originally. It can be seen that smaller k results in more IoT tasks being placed leftward after reordering. It further leads to shorter response time of IoT tasks and being harder to satisfy the QoS of common tasks. It should be noted that k literally has a range and can be easily calculated based on the inequality.

$$k \geq \frac{\sum_{x_{IoT}^{i} \in S} i}{\sum_{x^{i} \in S} i} \tag{8}$$

The proposed algorithm is IoT-aware and should consider the response time of them since they are usually more delay-sensitive than others [43]. One of the approaches is rendering all IoT tasks executing first. But too much adjustment will harm the original execution of non-IoT tasks. The mechanism of the proposed reorder process based on the constraint (8) can avoid over-adjusting and reduces the number of swapping places as less as possible. It can guarantee the overall or relatively superior execution order of IoT tasks, yet absolutely superior.

3.1.4 The Algorithm Procedure

The main ideas of main resource load balancing algorithm in heterogeneous environment and the one in non-heterogeneous environment are almost the same except two points. First, the former one will sort tasks and then reorder before assigning them to VMs. Second, the idea of relative load is introduced. Instead of the usage of $Normal(x, c_i)$, $Normal(x, c_i, vm_j)$ is used to represent the added c_i load of x upon VM j.

The procedure of main resource load balancing algorithm in heterogeneous environment is that firstly the tasks are sorted by their priorities in decreasing order. Then, reordering is performed based on the constraint (8). The sorting is pretty easy while the reordering needs a little trick.

To reduce the number of swapping as less as possible, two pointers are used to indicate the current swapping tasks, respectively. One of them starts from the beginning of the queue and probes towards the end of the queue. The other starts from the end of queue and probes towards the beginning. While the lower pointer is still lower than the higher, the lower pointer moves forward one step each time trying to locate the first non-IoT task and the higher pointer moves backward one index each time to locate the first last IoT task. After finishing the current swapping, the constraint (8) is checked. If it is satisfied, the loop ends. Otherwise, the loop continues. It should be noted that the validity of the given k is pre-checked and the value of the step is also pre-set by users. Only if the constraint is still not satisfied after the loop has ended, the value of the step will be decreased by one. And then another loop begins until the constraint is satisfied.

During the iteration of *CloudletList*, the unassigned task in the sorted and reordered queue is assigned sequentially to a properly selected VM. To be more

detailed, given a normal value of each type of resource, the algorithm calculates the normalized average demand of each task for each type of resource and the maximum normalized average demand of each task is regarded as the task's main resource load. Then, it iterates around all VMs and assigns the task to a VM with the minimum added load of resource *target* (also known as *Normal*(x, c_i, vm_j) here). The *target* denotes the to-be-assigned task's main resource. Finally, the VM which the current task is assigned to calculates its own new load by adding the newly arrived task's relative resource load to its previous load. If and only if a task finishes executing, the VM subtracts the task's relative resource load from its load. The pseudo-code of the algorithm is shown as Fig. 2.

It should be noted that *S*(*VmList*, *HostList*) denotes the set of hash mapping from VMs to hosts. Here we analyze the algorithm complexity. Supposed that there is N tasks and M VMs. So, the complexity of sorting tasks by their priorities based on our sorting algorithm is O(*N*logN)*. Then the calculation in Line 4 can be resolved in O(1) and greedily assign tasks to suitable VMs by iterating around tasks is O(M). Hence, for both the multi-resource scheduling algorithm for heterogeneous environment their algorithm complexity is O(*N*logN* + *N*M* + *N*).

1 **Input**: $CloudletList$, $VmList$, $c_1^*, c_2^*, \cdots, c_m^*$, α, k, $REFERENCE\_TIME$, $REFERENCE\_CPU$;

Output: Assignment of $CloudletList$;

2 Init $Load(vm_i, c_j) = 0, i = 1, \cdots, n,\ j = 1, \cdots, m$;

3 Sort $CloudletList$ by priority and then reorder them based on k;

3 **Foreach** cloudlet x in $CloudletList$

4 Calculate $Normal(x, c_1), \cdots, Normal(x, c_j)$, find the maximum $Normal(x, c_{target})$;

5 Find the vm_i with the minimum $Load(x, c_{target}, vm_i)$;

6 Assign cloudlet x to vm_i, $Load(vm_i, c_j) = Load(VM_i, c_{target}) + Load(x, c_{target}, VM_i), j = 1,2, \cdots, m$;

7 **return** the assignment of $CloudletList$.

Fig. 2 The main resource load balancing algorithm in heterogeneous environment

3.2 Time Balancing Algorithm

Another algorithm is also proposed, namely time balancing algorithm. Unlike the main resource load balancing algorithm, this one does not consider the factor of multi-resource. In fact, it is actually an extended version of Max-Min [44] based on MultiRECloudSim. Max means that the task with maximum priority is served first. Min means that the VM that can bring minimum execution time is selected first. And time balancing algorithm only takes task's estimated execution time (time cost) into account and greedily assigns tasks to VM with shortest total run time of tasks. In other words, the time cost is seen as a type of special load here. Therefore, the mechanism of this proposed method renders itself lack of some good qualities which the main resource load balancing has. It is only aimed to reduce make-span.

For all that, this algorithm considers the heterogeneity of VMs performance. It means that the same task will add different load to different VMs. If we use *Load*(x, vm_i) to denote the relative time cost of task x that is assigned to VM i, then we get the following two equations: Equations (9) and (10). Referring to these two equations, it is not difficult to find out the similarity between Max-Min and time balancing.

Also, time balancing algorithm integrates the aforementioned sorting and reordering technique.

$$Load(x, vm_i) = \frac{time(x)}{c_{cpu}(vm_i)} = \frac{length(x)}{c_{cpu}(x)c_{cpu}(vm_i)} \tag{9}$$

$$Load(vm_i) = \sum_{x \in E(i)} Load(x, vm_i) \tag{10}$$

Load(x, vm_i) here denotes the relative time cost of task x upon VM i. The pseudo-code of the algorithm is shown as Fig. 3.

Here we analyze the algorithm complexity. Supposed that there is N tasks and M VMs. So, the complexity of sorting tasks by their priorities based on our sorting algorithm is O(*N*logN*). Then greedily assigning tasks to suitable VMs by iterating around tasks is O(*M*). Hence, for both the multi-resource scheduling algorithm for heterogeneous environment their algorithm complexity is O(*N*logN* + *N*M*).

1**Input:** $CloudletList$, $VmList$, $c_1^*, c_2^*, \cdots, c_m^*$, α, k, $REFERENCE\_TIME$, $REFERENCE\_CPU$;

Output: Assignment of $CloudletList$

2 Init $Load(vm_i) = 0, i = 1, \cdots, n$;

3 Sort $CloudletList$ by Priority and then reorder them based on k;

3 **Foreach** cloudlet x in $CloudletList$

4 Find the vm_i with the minimum $Load(vm_i)$;

5 Assign the cloudlet x to vm_i, $Load(vm_i) = Load(vm_i) + Load(x, vm_i)$;

6 **return** Assignment of $CloudletList$.

Fig. 3 The time balancing algorithm in heterogeneous environment

4 Theory Analysis of the Proposed Algorithms

This section will perform theory analysis on main resource load balancing algorithm, Max-Min fairness algorithm [45] and time balancing algorithm in terms of weighed priority and relative load. It should be noted that the Max-Min fairness algorithm refers to a policy that successively assigns tasks decreasingly based on the run time to a VM with the shortest total run time after sorting them by their run time.

4.1 Weighed Sorting

In section 3.1 we have mentioned task's priority and the greater the priority is the earlier the task will be assigned. The priority is used to describe the degree of a task's impact on make-span of total tasks and the later the tasks with higher priorities being assigned will increase make-span. Obviously, if a task with long run time were arranged as the last finished one, there would exist relatively greater possibility of increasing make-span and vice versa. Besides, the tasks with more requested CPU are also relatively hard to assign. In all, we need to arrange the tasks with longer execution time and more requested CPU first.

4.1.1 The Readiness for the Analysis of the Priority Function

In Eq. (7), α denotes a value which is between [0,1]. It balances the relative importance between task estimated execution time and requested CPU. In this section, it was substituted by 0.7, the $REFERENCE\_TIME$ was substituted by 30, and $REFERENCE\_CPU$ was substituted by 900. Besides, through

$$time(x) = \frac{length(x)}{c_{cpu}(x)} \tag{11}$$

, there is:

$$Priority(x) = \frac{2length(x)}{75c_{cpu}(x)} + \frac{c_{cpu}(x)}{4500} \tag{12}$$

Therefore, after substitution, the Eq. (12) turns into $y = \frac{2L}{75x} + \frac{x}{4500}$, where $L \in [1000, 90000]$, $x \in [100, 2000]$. Here, L denotes the length of task and x denotes the requested demand of task. The figure of this bivariate function is shown as Fig. 4.

Based on Fig. 4, it is difficult to learn the relation between x and L. So, we can assume that L is 1000, 10,000, and 90,000, respectively. In this way, the original function turns into x-concerned only.

4.1.2 L = 1000

Based on Fig. 5, y is relatively small and generally smaller than 0.45 when L=1000. In interval [0,1], y is

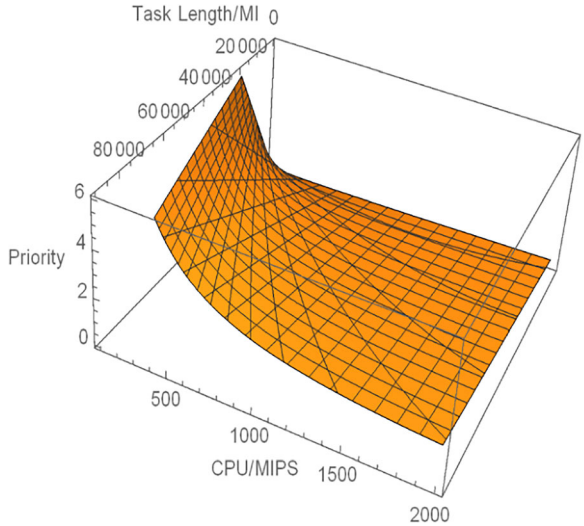

Fig. 4 2D image of the priority function

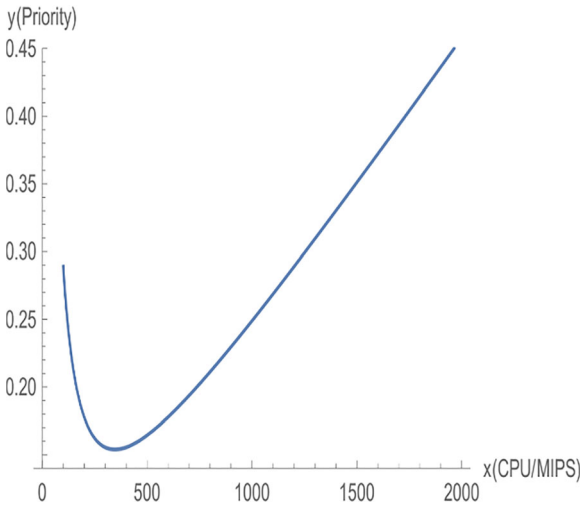

Fig. 5 Image of L = 1000

an increasing function of x which indicates that the shorter tasks have smaller priorities and the relation between CPU's demand and priorities is proportional.

4.1.3 L = 10,000

As shown in Fig. 6 in which L=10,000, there is a minimum value when x is approximately equal to 1100 and y has an interval of [0.49, 100,000]. When $x < 500$, there is $y > 0.6$. When $500 <= x <= 2000$, there is y in an interval of [0.49, 0.6]. Therefore, small x leads to high priority since the run time is long accordingly. With the increase of x, there appears an inflection point which indicates a balancing point of the relative importance between task run time and requested CPU. Run time is more important in the left side of inflection point and vice versa. In another word, if tasks are similar to each other in run time, tasks with more demand for CPU should be scheduled first.

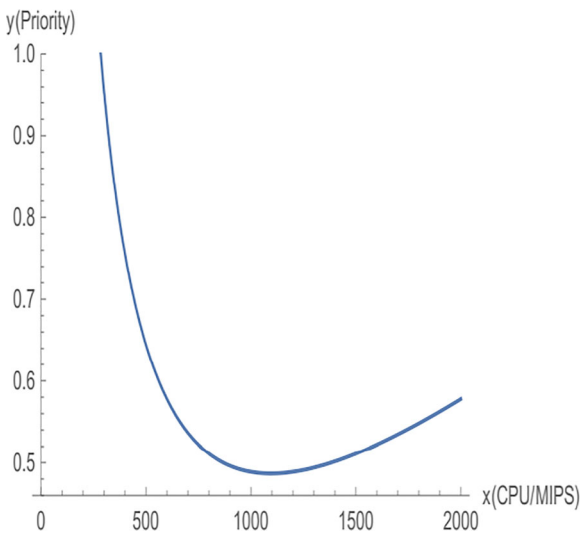

Fig. 6 Image of L = 10,000

4.1.4 L = 90,000

When L=90,000, as shown in Fig. 7, we can find that y is larger than 1 which means that y is generally large. Besides, y is the decreasing function of x which indicates that tasks with large length have high priority and when the length is large enough, the larger the demand of CPU is the less the run time is and the lower the priority is. Hence, the weighed priority takes tasks' run time and CPU demand into comprehensive consideration and sorts out tasks that need to be scheduled first dependent on different scenarios.

In the Max-Min fairness algorithm [45], the tasks will be sorted by estimated execution time only. That is to say, it has Eq. (13).

$$y = \frac{L}{30x} \tag{13}$$

The coefficient 1/30, in fact, can be spare. But it's added here for better comparison with the discussion of the weighed priority mentioned above. The number 30 denotes *REFERENCE _ TIME*. Its image is shown in Fig. 8 where L=10,000.

The drawback of this way to calculate priority is very obvious. For example, if there is a task named A with a length of 15,000 demanding 1000 MIPS, a task named

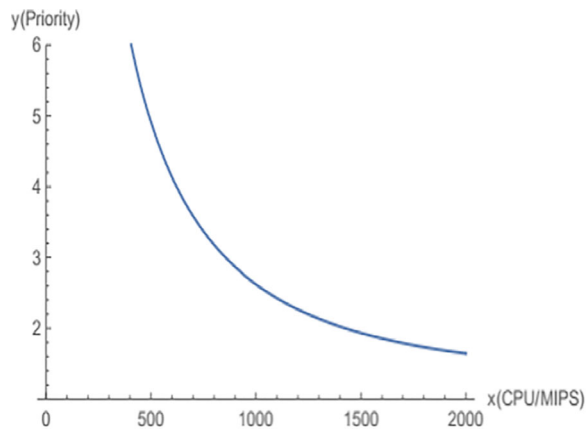

Fig. 7 Image of L = 90,000

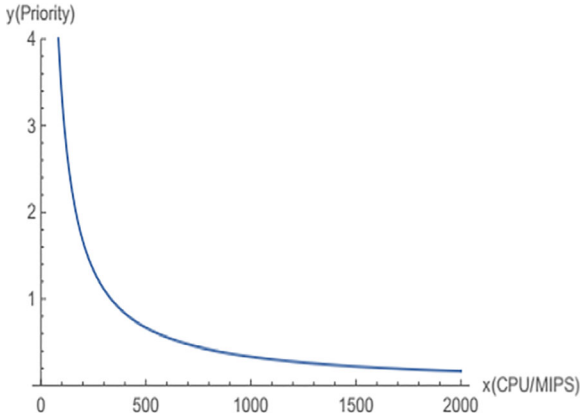

Fig. 8 Image of $y-\frac{10000}{30x}$

B with a length of 7500 demanding 500 MIPS and a task named C with a length of 28,000 demanding 2000 MIPS. It is easy to work out the run time of the three tasks are 15 S, 15 S, and 14 S, respectively. Therefore, in Max-Min fairness algorithm the relation among the priorities of the tasks is *Priority*(*A*) = *Priority*(*B*) > *Priority*(*C*). As for the weighted priority scheme, the result is different. Their priorities are 0.62, 0.51 and 0.81, respectively. Although the run time of task A and task B are the same, it is more difficult for scheduling because A demands more CPU. Hence, the priority of A should have been higher than B's. With respect to C whose run time is close to A and B's, it demands far more CPU than the previous two tasks'. The amount of the demand of tasks C for CPU occupies 65% of the workload of a CPU with a MIPS of 3GHz while A and B occupy 32% and 16%, respectively. Thus, C should be scheduled first. In conclusion, the weighted priority scheme is more suitable for defining the degree of a task's priority in scheduling.

4.2 Relative Load

When assigning tasks to VMs, a problem occurs if we take the demand for the resource as the metric of the load. Assuming that there are two VMs named A and B with CPU frequency of 3GHz and 2GHz, respectively. When the same situation of tasks assignment inflicts on A and B, respectively, there is still the left computing capability of 1GHz in A whose 33% of its CPU resource is not exploited while B is fully loaded. Hence, it's not a good idea for heterogeneous hosts to take the demand for the resource as the metric of the load.

It would be a better option to take the utilization of a host as the metric of the load aiming to measure the load of heterogeneous hosts more properly. The reasons are as follows. First, each task's demand for resource will be mapped to a value belongs to the interval of [0, 1]. Second, the utilization can show us the change of load brought by the same task to different hosts. For example, a task demanding 1GHz will bring the load of 33.3% to a one-core host with a frequency of 3GHz while 50.0% to a 2GHz one.

On the other hand, the occupying time of resource is also one of the most important factors. The longer it is, the higher the load of hosts is. And vice versa, the definition of the load of a host will be introduced below.

We can now define the load of a *vm* as this: during a time range of *t*, the load of a *vm* for resource c_i is $t * u\ (vm, c_i)$ if the utilization of a VM for resource c_i is $u\ (vm, c_i)$. In all, the load of a *vm* is denoted by $W\ (vm, c_i)$.

If there are m PEs each of whom possesses a frequency of *f* in each host. Then the utilization of a host is $\frac{\sum_i u(cpu_i)}{m}$ if each PE's utilization is $u\ (cpu_i)$, $(i = 1, \ldots, m)$. So, we get the load of a host for CPU denoted by $t\frac{\sum_i u(cpu_i)}{m}$.

The demand of a task *x* for i_{th} types of resource is denoted by $c_i(x)$. When *x* is assigned to a *vm*, the increasing load of the *vm* is denoted as follow:

$$\begin{aligned}\Delta W(vm, c_i) &= time(x)\frac{c_i(x)}{c_i(vm)} \\ &= \frac{length(x)}{c_{cpu}(x)}\frac{c_i(x)}{c_i(vm)}\end{aligned} \tag{14}$$

Specifically, the increasing load of CPU is denoted as follow:

$$\begin{aligned}\Delta W_1(vm, c_{cpu}) &= \frac{length(x)}{c_{cpu}(x)}\frac{c_{cpu}(x)}{c_{cpu}(vm)} \\ &= \frac{length(x)}{c_{cpu}(vm)}\end{aligned} \tag{15}$$

If taking the demand for resources as the metric of the load, which is what previous researchers usually did, the

increasing CPU load of a *vm* is shown as follow when the task x is assigned to it.

$$\begin{aligned}\Delta W_2(vm, c_{cpu}) &= time(x)c_{cpu}(x) \\ &= length(x)\end{aligned} \tag{16}$$

In the main resource load balancing algorithm for non-heterogeneous environment, the increasing CPU load of a VM is shown as follow when the task x is assigned to it. In Eq. (17), N_x^{cpu} is used to denote $Normal(x, c_{cpu})$.

$$\begin{aligned}\Delta W_3(vm, c_{cpu}) &= time(x)N_x^{cpu} \\ &= \frac{length(x)}{c_{cpu}(x)}\frac{c_{cpu}(x)}{c_i^*} \\ &= \frac{length(x)}{c_i^*}\end{aligned} \tag{17}$$

While in the main resource load balancing algorithm for heterogeneous environment, the relative increasing load of a VM becomes this one, as shown in Eq. (18), in which $Normal(x, c_{cpu}, vm)$ is replaced by $N_x^{cpu}(vm)$.

$$\begin{aligned}\Delta W_4(vm, c_{cpu}) &= time(x)N_x^{cpu}(vm) \\ &= \frac{length(x)}{c_{cpu}(x)}\frac{c_{cpu}(x)}{c_{cpu}(vm)} \\ &= \frac{length(x)}{c_{cpu}(vm)}\end{aligned} \tag{18}$$

Since Max-Min is performed based on selecting a task with maximum execution time and assign to the resource which gives minimum completion time [46], the increasing load of Max-Min fairness algorithm becomes the one shown in Eq. (19).

$$\begin{aligned}\Delta W_5(vm, c_{cpu}) &= time(x) \\ &= \frac{length(x)}{c_{cpu}(x)}\end{aligned} \tag{19}$$

With respect to the time balancing algorithm, it is shown in Eq. (20).

$$\begin{aligned}\Delta W_6(vm, c_{cpu}) &= Load(x, vm) \\ &= \frac{length(x)}{c_{cpu}(x)c_{cpu}(vm)}\end{aligned} \tag{20}$$

It can be seen from Table 3 that the scenario 1 is the reference definition, which takes the product of the resource amount and the resource occupation time as the resource load, given by us. As we have discussed

Table 3 The different calculation method of the load in different algorithms.

	$\Delta W(vm, c_i)$	CPU	Non-CPU
1	Reference	$\frac{length(x)}{c_{cpu}(vm)}$	$\frac{length(x)}{c_{cpu}(x)} \times \frac{c_i(x)}{c_i(vm)}$
2	Taking the demand for resources as the metric of the load	$length(x)$	$\frac{length(x)}{c_{cpu}(x)} \times c_i(x)$
3	The main resource load balancing algorithm for non-heterogeneous environment	$\frac{length(x)}{c_{cpu}^*}$	$\frac{length(x)}{c_{cpu}(x)} \times \frac{c_i(x)}{c_i^*}$
4	The main resource load balancing algorithm for heterogeneous environment	$\frac{length(x)}{c_{cpu}(vm)}$	$\frac{length(x)}{c_{cpu}(x)} \times \frac{c_i(x)}{c_i(vm)}$
5	The Max-Min fairness	$\frac{length(x)}{c_{cpu}(x)}$	
6	The time balancing	$\frac{length(x)}{c_{cpu}(x)c_{cpu}(vm)}$	

before, it's impossible to balance the resource heterogeneity existing among hosts if taking the amount of the demand for resource as the load definition as demonstrated in scenario 2. Scenario 3 is similar to scenario 2 except for the fact that the normal values of different resources are introduced to eliminate the dimensionality between all resources. The normal values are fixed so it cannot reflect the different configuration among different hosts. Scenario 4 is closest to scenario 1 but it overcomes the shortcoming of the fixed normal values in scenario 3. The key lies in the use of the coefficient 1/m in which the m denotes the number of PEs or VMs. But if the changes of load refer to hosts it will still be the same. Its purpose is to consider the issue from the perspective of the host. It's designed to optimize hosts rather than VMs. None of scenario 5 and 6 considers the factor of multi-resource scheduling and they only focus on developing the scheduling scheme based on single dimension factor which is the task completion time. That may lead to high efficiency by the two algorithms. In contrast, scenario 6 outperforms scenario 5 in efficiency since it takes the heterogeneity among VMs into account. However, the main resource load balancing algorithm for heterogeneous environment might have advantages in resource load balancing and energy consumption.

5 Experiments and Results

All the evaluations are performed on the platform of MultiRECloudSim. The simulation settings are listed in Tables 4, 5, 6, and 7.

In Table 4, some global settings and the class we used are listed. We assumed that there was only data-center in Cloud. We could have increased the number of data-centers but it was unnecessary and redundant for explaining the proposed ideas. Therefore, the number of hosts remained unchanged during the execution process. Since each host has a fixed number of VMs (according to Table 5), the total number of VMs also remained unchanged.

As we have mentioned that the proposed ideas try to avoid VM migration, PowerVmAllocationPolicyByHost was used as the global setting as well. This policy does not consider the run-time VM migration and will arrange VMs based on the current host type. It should be noted that the policy was written by the authors of CloudSim. The rest in Table 4 mostly was developed by the authors of MultiRECloudSim, except for SimProgressCloudletIoT and SimPowerHostMultiR. The SimProgressCloudletIoT was extended for indicating that the current task is an IoT task or a Common task.

According to Tables 5 and 6, it is not difficult to find that the each type of VM's configurations are averaged based on the configurations of the corresponding type of host. That is because we reasonably assumed that each VM can only have one PE. In this way, the VM allocation process is much more simplified and we can pay more attention to the working of task scheduling. The VM configurations can be much more complex than what are shown in Table 6 but still it is unnecessary and redundant.

As seen in Table 7, there are three major types of tasks. We synthesized the tasks based on the related article [47] and stored them into a file. The ranges mean that the values were selected randomly within the ranges. Each task's configurations are separated by a tab space and stored as a whole in a line of the file. In the end, in order to differentiate the IoT tasks and the Common tasks, the labelling process must be done based on a rule. The rule is that the chosen lines in the file will be labelled as IoT while the rest lines will be labelled as Common. During the creation of SimProgressCloudletIoT, those information will be used to set up the cloudlets.

In this section, six different algorithms were used to perform experiments to verify the effectiveness of the proposed algorithms, including first come first serve (Algo1), random assignment (Algo2), trade-off (Algo3) [36], main resource load balancing (Algo4), time balancing (Algo5), and main resource task balance

Table 4 Global settings

Parameters or class	Value
Interval	5 s
Number of hosts	40
Number of VMs	340
Number of tasks	8000
VmScheduler	SimVmSchedulerTimeSharedOverSubscription
CloudletScheduler	SimCloudletSchedulerDynamicWorkload
AllocationPolicy	PowerVmAllocationPolicyByHost
Cloudlet	SimProgressCloudletIoT
VM	SimPowerVm
Host	SimPowerHostMultiR
Broker	SimDatacenterBroker
Datacenter	SimPowerDatacenter
CPU/Ram/IoAllocator	CPU/Ram/IoAllocatorSimple
Cloudlet resource utilization	UtilizationProgressModelByFile
PeProvisioner	PeProvisionerSimple
Ram/IoProvisioner	SimRam/IoProvisionerSimple

Table 5 Host configuration parameters

Parameters	Type one	Type two	Type three	Type four
Number of PEs	4	6	8	16
Frequency of each PE	2933 MHz	3067 MHz	2048 MHz	3500 MHz
RAM	4 GB	12 GB	6 GB	16 GB
IO	300 MB/s	500 MB/s	400 MB/s	300 MB/s
CPU power model	PowerModelSpecPowerI bmX3550XeonX5675	PowerModelSpecPowerHpProLiant Ml110G5Xeon3075	PowerModelSpecPowerHpProLiant Ml110G3PentiumD930	PowerModelSpecPowerHpProLiant Ml110G5Xeon3075
RAM power model	PowerModelRamSimple			
IO power model	PowerModelIoSimple			

(Algo6) [42], and. It should be noted that TB is actually an extended version of Max-Min based on MultiRECloudSim and TO was implemented based on the idea of [36]. In Table 8, some algorithms' particular parameters are presented.

5.1 Evaluation on the Awareness of IoT

In this section, the response time of IoT tasks by different algorithms will be evaluated. To be more detailed, the task's finish time is regarded as the corresponding response time. As we all know, the network delay is unavoidable and sometimes out of the cloud provider's control. But it is also fair for all the submitted tasks. Therefore, if we want to know how well different algorithms' response the request from IoT devices, we should just compare the task finished time.

Figure 9 illustrates the overall cloudlets' response time produced by different algorithms. It is not difficult to find that Algo5 has the minimum maximum response time and the minimum response time. The symbol on each box's middle stands for the average value.

In Table 9, the detailed values are listed. In fact, both Algo4 and Algo5 perform well in term of the response time of IoT tasks. But Algo4 performs 19.97% worse than Algo5. That's because compared to Algo4, Algo5 only considers task execution time when assigning tasks.

5.2 Evaluation on the Host SLA

Figure 10 shows the comparison of part of hosts' SLA results. The SLA is calculated by the ratio of the total service violation time and the total execution time. In CloudSim, the host will update its state history every time interval. And the current requested resources and allocated resources are recorded in the state history entry. Therefore, it is not difficult to get the SLA result.

Due to the limited space of the chart, only part of hosts is presented in the figure. It can be seen that the SLA produced by Algo4 is either the lowest or the second lowest. To be more convincible, detailed analysis is listed in Table 10. In all, the SLA result produced by Algo4 is the best.

While Algo4 performs well in term of host SLA, Algo5 performs 42.54%, 79.70%, and 20.91% worse than Algo3, Algo4, and Algo6, respectively. The reason is simple and has been mentioned at the end of section 5.1.

Table 6 VM configuration parameters

Parameters	Type one	Type two	Type three	Type four
Number of VMs in each corresponding host type	4	6	8	16
Frequency	2933 MHz	3067 MHz	2048 MHz	3500 MHZ
RAM	1 GB	2 GB	0.75 GB	1 GB
IO	75 MB/s	83 MB/s	50 MB/s	18.75 MB/s

Table 7 Task parameters

	Length	CPU (MHz)	RAM (MB)	IO (MB/s)	Quantity ratio
Type one	5655–110,157	800–1600	256	5	2
Type two		100–800	384–896	5	1
Type three		100–800	256	20–60	1

5.3 Evaluation on the Degree of Load Balance

To evaluate the degree of load balance produced by different algorithms, we use the variance of all VMs' resource utilization as the degree of load balance. In Figs. 11, 12, and 13, the runtime changing of VMs' utilization are presented. Each type of colors stands for a VM and the numbers in the legend represent VMs' ID. Due to the limited space of the article, only Algo4's part of VMs' runtime results are presented As can be seen in Figs. 11, 12, and 13, the runtime utilization changed a lot and unstable. Therefore, we decide to use the average of utilization of runtime VM to stand for the VM's resource utilization.

In Table 11, the comparison of degree of load balance produced by different algorithms are presented. Since multi-resource scheduling is supported on MultiRECloudSim, three types of resource are considered in the comparison. It can be seen that the CPU's degree of load balance produced by Algo4 are much better that the others. The other types of resource's degree of load balance are not the best but still are not bad. Compared with Algo3, Algo3 is 8.05% better than Algo4 in RAM balance. Compared with Algo1, Algo1 is 1.18% better than Algo4 in IO. But neither Algo1 nor Algo3 is better than Algo4 in terms of SLA and IoT task response time.

At last, the overall degree of each algorithm is provided in Table 11. Without any doubt, Algo4 performs best in load balancing out the 6 algorithms.

While Algo4 performs well in term of load balance, Algo5 performs 26.64%, 71.16%, and 22.06% worse than Algo3, Algo4, and Algo6, respectively.

Table 8 Algorithm parameters

Parameters	MRLB/MRTB/TB
Normal value of the length	1000
Normal value of CPU	3067
Normal value of RAM	1024
Normal value of IO	83
Time weight in sorting (α)	0.7
CPU weight in sorting	0.3
k	0.2
Reference value of time for priority	30
Reference value of CPU for priority	900

5.4 Evaluation on the Make-Span

The make-span of the algorithm is obtained via subtracting the simulation start time from the simulation stop time. As can be seen from Fig. 14, Algo4 no longer stands for the best. Instead, it costs 31.81%, 27.60%, and 10.38% more time than Algo3, Algo5, and Algo6 do, respectively.

While Algo4 performs bad in term of make-span, Algo5 saves 21.62% and 13.49% more time than Algo4 and Algo6 do, respectively. It only cost 3.3% more time than Algo3 does.

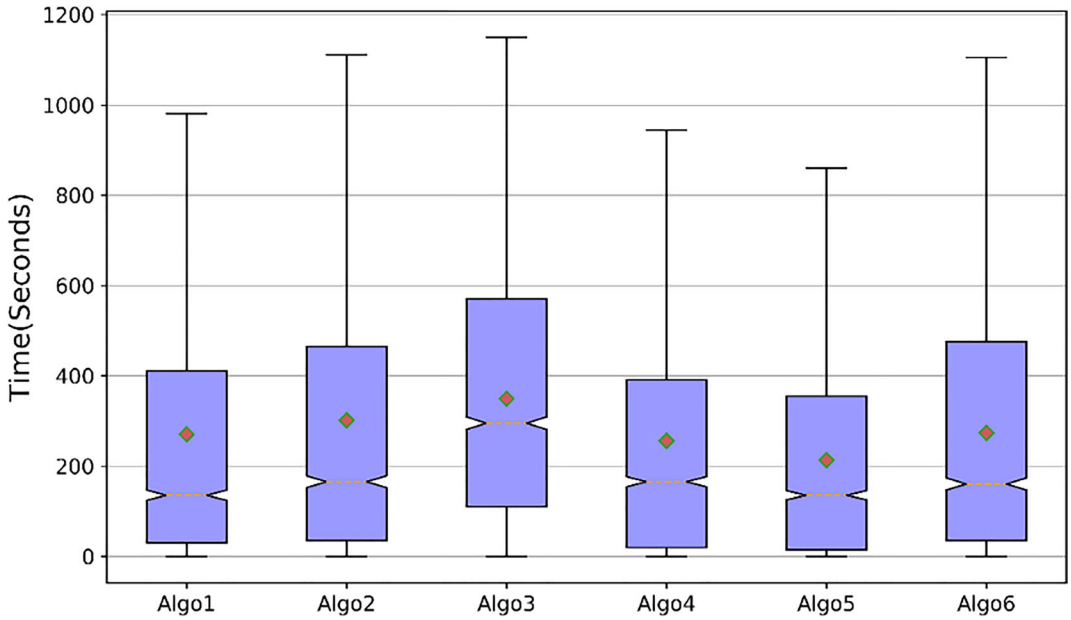

Fig. 9 The comparison of algorithms' IoT task response time

In fact, those results are not difficult to understand. Just as what had been mentioned above, Algo5 is actually a variant of Max-Min and it only considers how to shorten the execution time cost as much as possible. Yet, Algo4 considers more than that and it can only achieve the trade-off between make-span and the others to some extent.

5.5 Evaluation on the Energy Consumption

As can be seen from Fig. 15, Algo4 still is the worst among the last four algorithms. It generates 21.24%, 20.64%, and 5.97% more energy than Algo3, Algo5, and Algo6 do, respectively. Those result are also easy to explain. In CloudSim or some other similar cloud simulation platforms, if the simulation settings and the hardware specifications remain unchanged, the power consumption also remains unchanged. Therefore, in general, the longer the make-span or execution time is, the more the energy is consumed.

5.6 Evaluation on the k

To evaluate what impacts will be produced via changing k, we should fix other experiment settings mentioned above. It should be noted that this evaluation is of qualitative analysis rather than of quantitative analysis. It aims to tell the differences which is caused by different k.

For simplicity and also for generality, Algo4 is used to evaluate. The same impacts will also be observed from Algo5 because k only influences the process of deciding tasks' places in the task queue. And both Algo4 and Algo5 have the same process. Here, 6 different values are provided for evaluation on k, as shown in the legend of Fig. 16.

Table 9 The comparison of algorithms' IoT average task response time

	Algo1	Algo2	Algo3	Algo4	Algo5	Algo6
Average response time (Seconds)	270.19	293.65	367.63	255.74	213.17	273.45

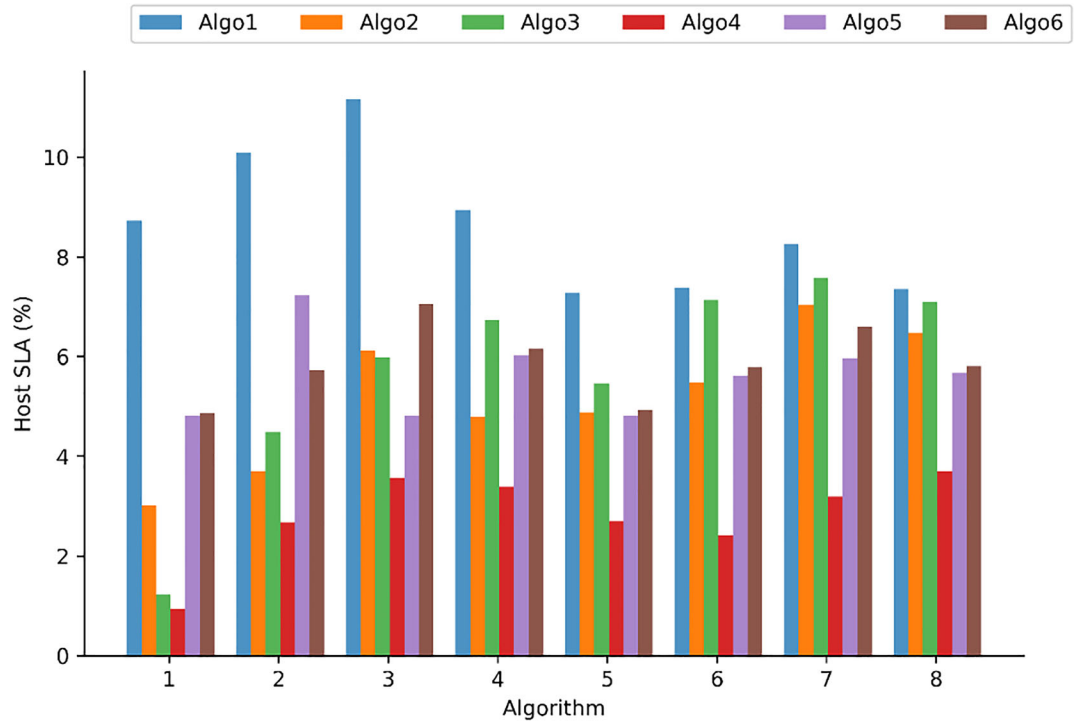

Fig. 10 The comparison of algorithms' host SLA

In Fig. 16, it can be seen that the SLA results are the same when k=0.1, k=0.2, k=0.3, k=0.4, k=0.5, and k=0.6. The reason is that k has its own legal range. This can be proved according to the constraint (8). The range is decided by the number of IoT tasks and the number of all kinds of tasks. Besides, we have also mentioned that in the section 3.1. To original statement is that "It should be noted that the validity of the given k is pre-checked and the value of the step is also pre-set by users". Therefore, if k is out of legal range, it will not influence the algorithms any more.

Based on this result, we decide to modify the values to (0.2, 0.22, 0.24, 0.26, 0.28, 0.3). The new results are shown in Table 13. It can be seen from the table that the relation between SLA and k is not monotonous. There is a minimum SLA point between 0.2 and 0.24.

There are also evaluations on IoT task response time, load balance, energy consumption, and make-span. They are all presented in Tables 12 and 13. About the IoT task response time, the relation is undoubtedly monotonically increasing. Because greater k means more IoT tasks are placed behind. About load balance, the relation between load balance and k is not monotonous. There is an optimal load balance point between 0.2 and 0.24. About energy consumption and make-span, they are more unstable than the other parameters when k changes. As shown in Fig. 17, the values slightly fluctuate around a line. Therefore, there make-span and energy consumption have no direct relation with k.

Table 10 The comparison of algorithms' average host SLA

	Algo1	Algo2	Algo3	Algo4	Algo5	Algo6
Average of all hosts' SLA (%)	8.14	6.52	5.03	3.99	7.17	5.93

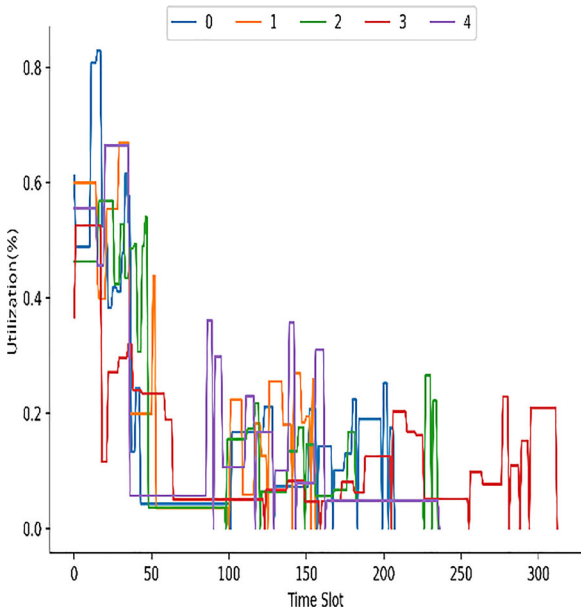

Fig. 11 Part of VMs' CPU utilization of Algo4

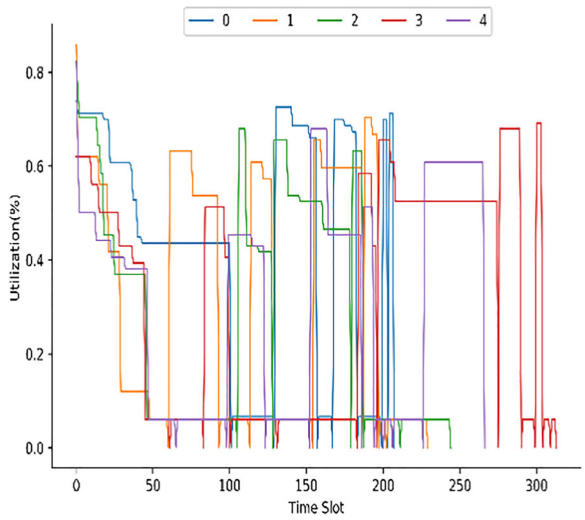

Fig. 13 Part of VMs' IO utilization of Algo4

5.7 Evaluation on the α

To evaluate what impacts will be produced via changing α, we should fix other experiment settings mentioned above. It should be noted that this evaluation is also of qualitative analysis rather than of quantitative analysis. It aims to tell the differences which is caused by different α.

Likewise, Algo4 is used to evaluate. The same impacts will also be observed from Algo5 because α only influences the process of deciding tasks' places in the task queue. And both Algo4 and Algo5 have the same process. Here, α is set to be 0.2, 0.3, 0.4, 0.5, 0.6, and 0.7 during the experiments, respectively.

In Table 14, IoT task response time, load balance, energy consumption, SLA, and make-span are all presented. It is not difficult to find that IoT task response time, load balance, and SLA do not have regular relations with α. But the energy consumption and make-span have. According to Fig. 18, with the growth of α, the general trend of both energy consumption and make-

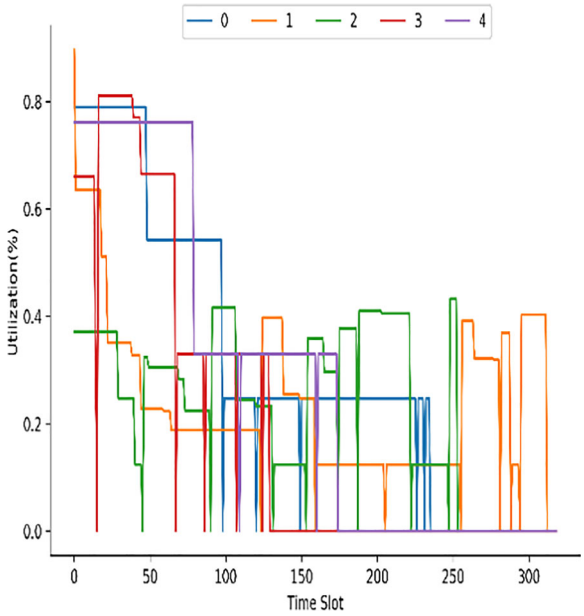

Fig. 12 Part of VMs' RAM utilization of Algo4

Table 11 The comparison of algorithms' degree of load balance (the smaller the better)

	Algo1	Algo2	Algo3	Algo4	Algo5	Algo6
CPU	18.33	11.94	7.76	2.21	10.54	6.73
RAM	24.85	21.55	8.67	9.43	13.29	10.87
IO	12.46	13.37	16.32	12.61	17.65	16.40
Overall	18.54	15.62	10.92	8.08	13.83	11.33

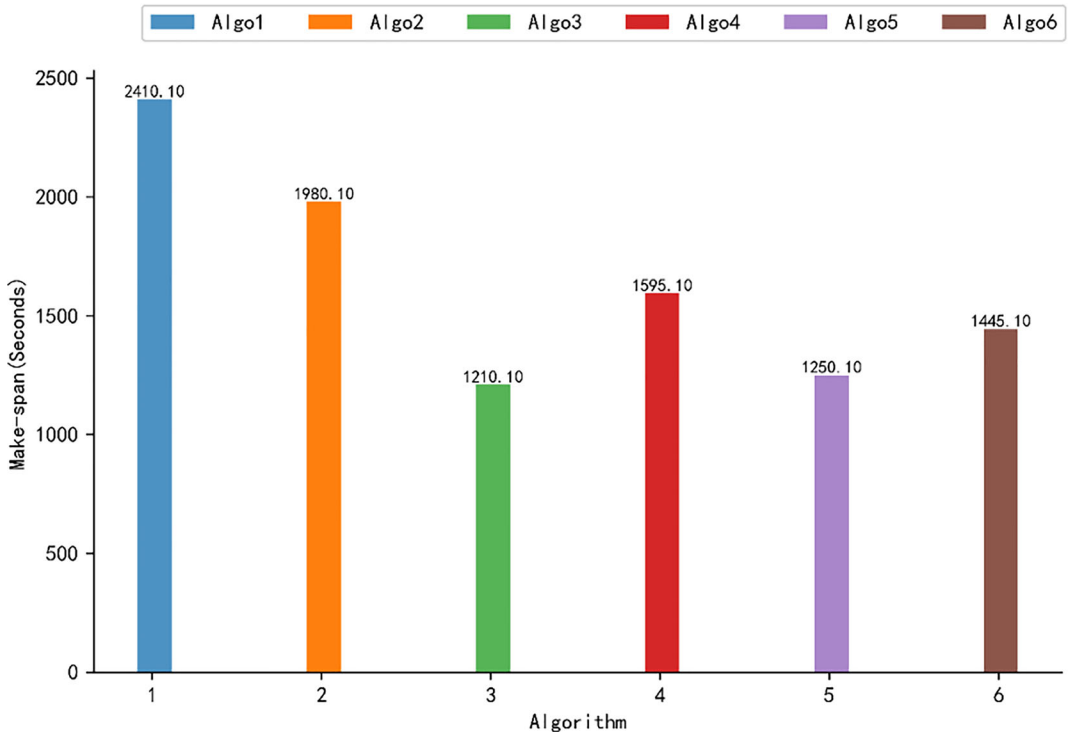

Fig. 14 The comparison of algorithms' make-span

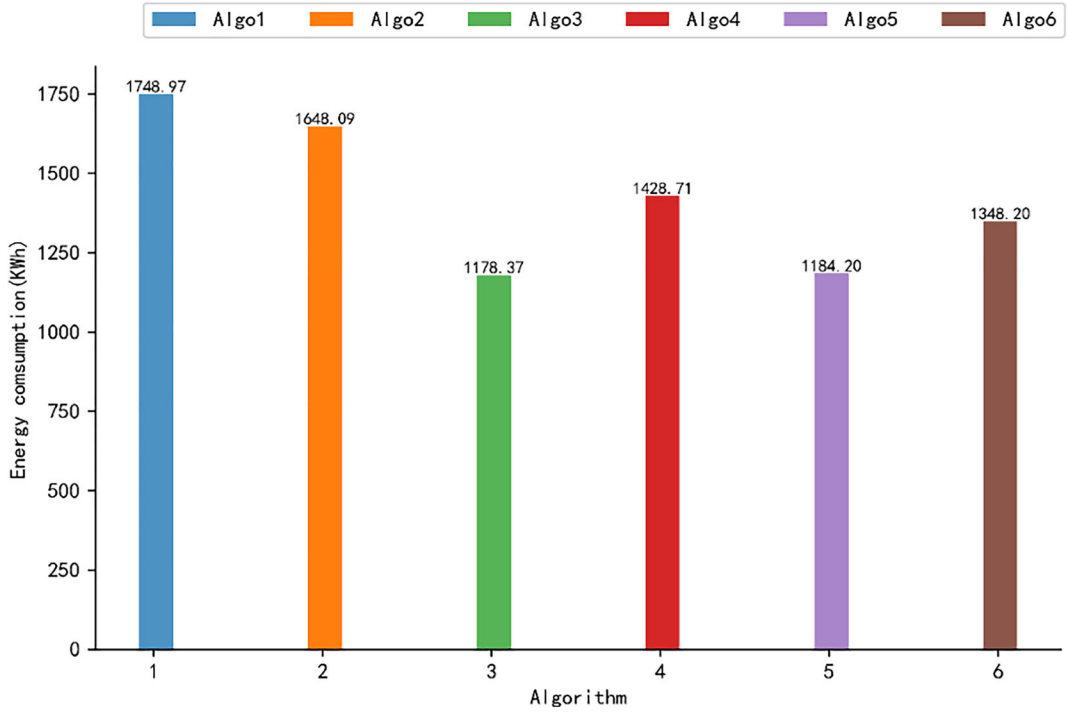

Fig. 15 The comparison of algorithms' energy consumption

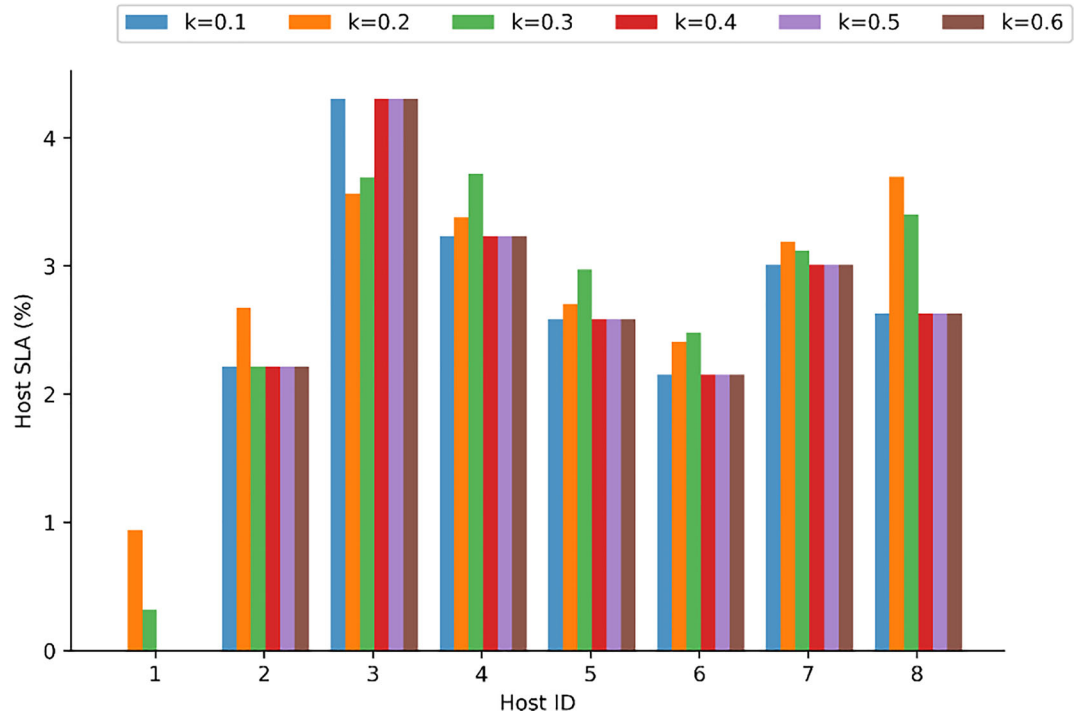

Fig. 16 Different *k*'s impacts on the host SLA of Algo4

Table 12 The comparison of algorithms' energy and make-span

	Algo1	Algo2	Algo3	Algo4	Algo5	Algo6
Energy (KWh)	1748.97	1648.09	1178.37	1428.71	1184.20	1348.20
Make-span (seconds)	2410.10	1980.10	1210.10	1595.10	1250.10	1445.10

Table 13 The comparison of different *k*'s impacts

	k=0.2	*k*=0.22	*k*=0.24	*k*=0.26	*k*=0.28	*k*=0.3
Average response time (Seconds)	255.74	273.39	308.41	324.37	339.98	373.49
Average of all hosts' SLA (%)	3.994	2.5809	3.224	3.438	3.486	3.694
Energy (KWh)	1428.71	1439.75	1427.31	1399.51	1414.41	1399.99
Make-span (seconds)	1595.10	1660.10	1600.10	1605.10	1640.10	1585.10
Degree of CPU balance	2.215	2.350	2.146	2.150	2.079	2.019
Degree of RAM balance	9.413	11.063	10.630	10.378	10.045	9.396
Degree of IO balance	12.692	13.429	12.642	11.697	11.438	11.076
Overall balance	8.107	8.947	8.473	8.075	7.854	7.497

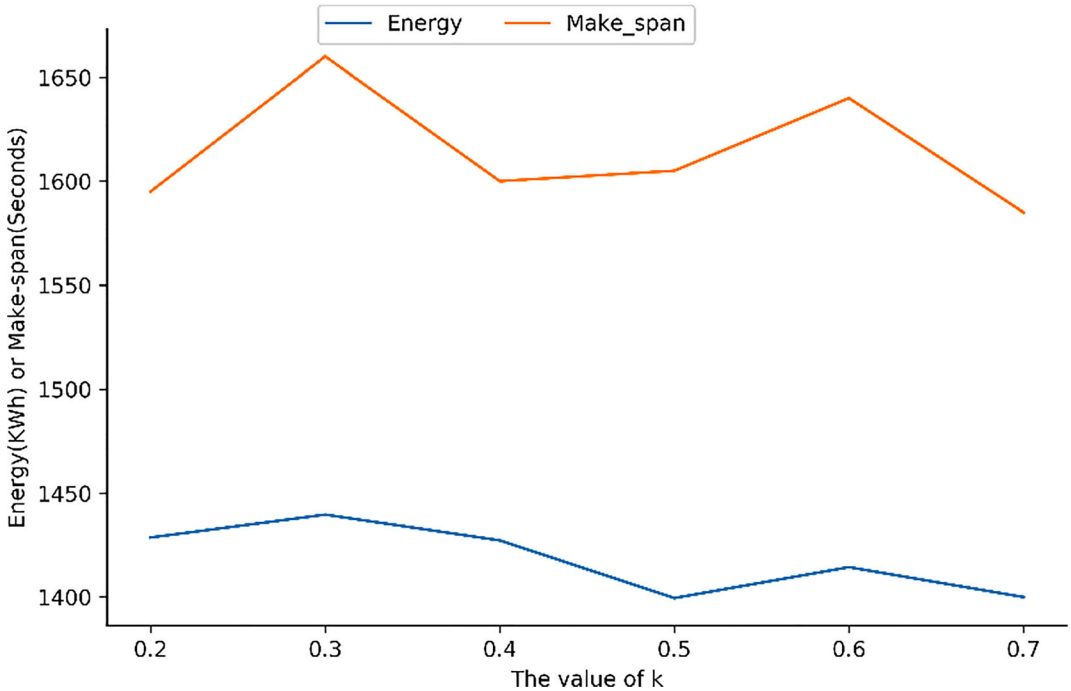

Fig. 17 The impacts on energy and make-span by different *k*

span are decreasing. It is easy to explain such situations. As what has been mentioned in section 5.5, longer make-span mean more energy is consumed. And in the second part of the section 3.1, we have demonstrated that "the priority of a task is only determined by estimated execution time when α=1". Therefore, the greater α is, the more time the algorithms will save. This result fully proves that the tuning of α does control the behavior of the algorithms.

6 Conclusion and Future Work

Two novel IoT-aware multi-resource task scheduling algorithms are presented, aiming to achieve better results in load balancing, SLA, and IoT task response time while producing trade-offs between make-span and energy consumption to some extent. They sequentially assign tasks from the sorted and reordered task queue to a properly selected VM. Under the control of two

Table 14 The comparison of different α's impacts

	α=0.2	α=0.3	α=0.4	α=0.5	α=0.6	α=0.7
Average response time (Seconds)	256.48	251.57	255.33	252.42	253.02	255.74
Average of all hosts' SLA (%)	2.96	2.51	3.89	3.12	3.08	3.99
Energy (KWh)	1493.88	1480.77	1479.95	1467.92	1473.94	1428.71
Make-span (seconds)	1865.10	1740.10	1725.10	1725.10	1720.10	1595.10
Degree of CPU balance	2.226	2.083	2.251	1.999	2.357	2.215
Degree of RAM balance	10.129	8.711	9.512	9.427	9.618	9.413
Degree of IO balance	12.394	11.435	12.525	11.972	12.130	12.692
Overall balance	8.250	7.410	8.096	7.799	8.035	8.107

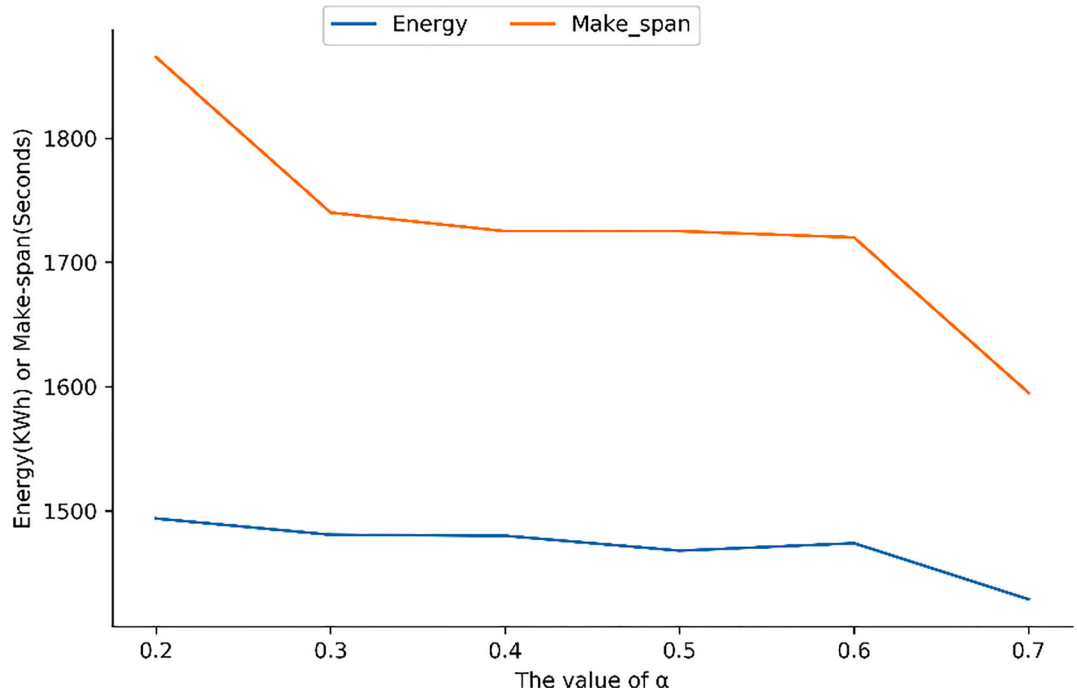

Fig. 18 The impacts on energy and make-span by different α

parameters, the pre-processed task queue is produced. In the main resource load balancing algorithm (MRLB), the VM is chosen based on the least relative load. While in the time balancing algorithm (TB), the VM is chosen based on the least cost of time. This distinction results in different outcomes of energy consumption and make-span.

With the rise of Cloud-IoT, our algorithms address the growing need of optimizing QoS and load balancing while guaranteeing energy consumption to some extent. They also provide two parameters for engineers from cloud background to flexibly control the behaviors of the algorithms. Using the advanced version of simulation platform modified from the well-known CloudSim, our findings are evaluated against two traditional and two recently developed related approaches.

We will extend our work into the field of online and dependent task scheduling. It means that the relationship or dependency between tasks should be considered in the pre-processing of task queue. Future work avenues might aim to address the application scenarios of the much more light-weight containers and incorporate the approaches of workload prediction for resizing containers.

Acknowledgements This work is supported by National Natural Science Foundation of China (Grant Nos. 61772205, 61872084), Guangdong Science and Technology Department (Grant No. 2017B010126002), Guangzhou Science and Technology Program key projects (Grant Nos. 201802010010, 201807010052, 201902010040 and 201907010001), Guangzhou Development Zone Science and Technology(Grant No. 2018GH17), Special Funds for the Development of Industry and Information of Guangdong Province (big data demonstrated applications) in 2017, and the Fundamental Research Funds for the Central Universities, SCUT(Grant No. 2019ZD26).

References

1. Gill, S.S., Buyya, R.: Resource provisioning based scheduling framework for execution of heterogeneous and clustered workloads in clouds: from fundamental to autonomic offering. J Grid . Comput. 1–33 (2018)
2. Bera, Samaresh, Sudip Misra, and Joel JPC Rodrigues. "Cloud computing applications for smart grid: A survey." IEEE Transactions on Parallel and Distributed Systems **26**.5 (2014): 1477-1494.
3. Mell P., Grance T.: The NIST definition of cloud computing[J]. (2011)
4. Baker, T., et al.: Energy efficient cloud computing environment via autonomic meta-director framework. 2013 Sixth International Conference on Developments in eSystems Engineering. IEEE. (2013)

5. Baker, T., et al.: GreeDi: an energy efficient routing algorithm for big data on cloud. Ad Hoc Netw. **35**, 83–96 (2015)
6. Baker, T., et al.: Greeaodv: an energy efficient routing protocol for vehicular ad hoc networks. International Conference on Intelligent Computing. Springer, Cham. (2018)
7. Rathore, M.M., et al.: Urban planning and building smart cities based on the internet of things using big data analytics. Comput. Netw. **101**, 63–80 (2016)
8. Botta, A., et al.: On the integration of cloud computing and internet of things. 2014 international conference on Future internet of things and cloud (FiCloud). IEEE. (2014)
9. Zhao, X., Zhao, L., Liang K.: An energy consumption oriented offloading algorithm for fog computing. In: International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, Springer, pp. 293–301 (2016)
10. Hasan, R., Hossain, M., Khan, R.: Aura: an incentive-driven ad-hoc IoT cloud framework for proximal mobile computation offloading. Future Gener. Comput. Syst. (2017)
11. Shiraz, M., et al.: Energy efficient computational offloading framework for mobile cloud computing. J. Grid. Comput. **13**(1), 1–18 (2015)
12. Deshmukh, S., Shah, R.: Computation offloading frameworks in mobile cloud computing: a survey. 2016 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC). IEEE (2016)
13. Moreno, I.S., et al.: Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. IEEE T. Cloud. Comput. **2**(2), 208–221 (2014)
14. Gutierrez-Garcia, J.O., Ramirez-Nafarrate, A.: Agent-based load balancing in cloud data centers. Clust. Comput. **18**(3), 1041–1062 (2015)
15. Bala, M.: Proportionate resource utilization based VM allocation method for large scaled datacenters. Int. J. Inf. Technol. **10**(3), 349–357 (2018)
16. Xie, Lei, et al. "A Novel Self-Adaptive VM Consolidation Strategy Using Dynamic Multi-Thresholds in IaaS Clouds." Future Internet **10**.6 (2018): 52.
17. Kaur, A., Kalra, M.: Energy optimized VM placement in cloud environment. 2016 6th International Conference-Cloud System and Big Data Engineering (Confluence). IEEE (2016)
18. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Futur. Gener. Comput. Syst. **28**(5), 755–768 (2012)
19. Kansal, N.J., Chana, I.: Energy-aware virtual machine migration for cloud computing-a firefly optimization approach. J. Grid. Comput. **14**(2), 327–345 (2016)
20. Singh, G., Gupta, P.: A review on migration techniques and challenges in live virtual machine migration[C]//2016 5th international conference on reliability, Infocom technologies and optimization (trends and future directions) (ICRITO). IEEE, 542–546 (2016)
21. Boutaba, Raouf, Qi Zhang, and Mohamed Faten Zhani. "Virtual machine migration in cloud computing environments: Benefits, challenges, and approaches." Communication Infrastructures for Cloud Computing. IGI Global, 2014. 383-408.
22. Zhao, H., et al.: Power-aware and performance-guaranteed virtual machine placement in the cloud. IEEE T. Parall. Distr. **29**(6), 1385–1400 (2018)
23. Mohapatra, S, Majhi, B.: An evolutionary approach for load balancing in cloud computing. Handbook of research on securing cloud-based databases with biometric applications. IGI Global, 433–463 (2015)
24. Mondal, R. K., et al.: Load balancing of unbalanced matrix problem of the sufficient machines with min-min algorithm. Methodologies and application issues of contemporary computing framework, pp. 81–91. Springer, Singapore (2018)
25. Malik, A., Chandra, P.: Priority based round robin task scheduling algorithm for load balancing in cloud computing. Journal of Network Communications and Emerging Technologies (JNCET) www. jncet. org 7(12) (2017)
26. Mittal, S., Katal, A.: An optimized task scheduling algorithm in cloud computing. 2016 IEEE 6th International Conference on Advanced Computing (IACC). IEEE (2016)
27. Adhikari, M., Amgoth, T.: Heuristic-based load-balancing algorithm for IaaS cloud. Futur. Gener. Comput. Syst. **81**, 156–165 (2018)
28. Alaei, N., Safi-Esfahani, F.: RePro-active: a reactive–proactive scheduling method based on simulation in cloud computing. J. Supercomput. **74**(2), 801–829 (2018)
29. Singh, S., Chana, I.: A survey on resource scheduling in cloud computing: issues and challenges. J. Grid. Comput. **14**(2), 217–264 (2016)
30. Panda, S.K., Jana, P.K.: SLA-based task scheduling algorithms for heterogeneous multi-cloud environment. J. Supercomput. **73**(6), 2730–2762 (2017)
31. Zhou, J., Yao, X.: Multi-objective hybrid artificial bee colony algorithm enhanced with Lévy flight and self-adaption for cloud manufacturing service composition. Appl. Intell. **47**(3), 721–742 (2017)
32. Shojafar, M., Javanmardi, S., Abolfazli, S., et al.: FUGE: a joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method[J]. Clust. Comput. **18**(2), 829–844 (2015)
33. Asghari, S., Navimipour, J. N.: Cloud services composition using an inverted ant colony optimization algorithm. Int. J. Bio-Inspired Comput. (2017, in press) (2017)
34. Beheshti, Z., Shamsuddin, S.M.H.: A review of population-based meta-heuristic algorithms. Int. J. Adv. Soft Comput. Appl. **5**(1), 1–35 (2013)
35. Aazam, M., Zeadally, S., Harras, K.A.: Offloading in fog computing for IoT: review, enabling technologies, and research opportunities. Futur. Gener. Comput. Syst. **87**, 278–289 (2018)
36. Yang, J., Xu, X., Tang, W., et al.: A task scheduling method for energy-performance trade-off in Clouds[C]. 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE. IEEE, 1029–1036 (2016)
37. Chen, Congyang, et al. "Research on workflow scheduling algorithms in the cloud." International Workshop on Process-Aware Systems. Springer, Berlin, Heidelberg, 2014.
38. Juarez, F., Ejarque, J., Badia, R.M.: Dynamic energy-aware scheduling for parallel task-based application in cloud computing. Futur. Gener. Comput. Syst. **78**, 257–271 (2018)

39. Hussain, A., et al.: RALBA: a computation-aware load balancing scheduler for cloud computing. Clust. Comput. **21**(3), 1667–1680 (2018)
40. Calheiros, R.N., Ranjan, R., Beloglazov, A., et al.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software Pract. Exper. **41**(1), 23–50 (2011)
41. Lin, W., Xu, S., Li, J., et al.: Design and theoretical analysis of virtual machine placement algorithm based on peak workload characteristics[J]. Soft. Comput. **21**(5), 1301–1314 (2017)
42. Lin, W., Xu, S., He, L., et al.: Multi-resource scheduling and power simulation for cloud computing[J]. Inf. Sci. (2017)
43. Flores, H., et al.: Large-scale offloading in the Internet of Things. 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). IEEE (2017)
44. Panda, S.K., Agrawal, P., Khilar, P.M., Mohapatra, D.P.: Skewness-based min–min max–min heuristic for grid task scheduling. In: Proceedings of the 2014 Fourth International Conference on Advanced Computing & Communication Technologies, pp. 282–289 (2014)
45. Alharbi, F., Rabigh, K.S.A.: Simple scheduling algorithm with load balancing for grid computing[J]. Asian Transactions on Computers. **2**(2), 8–15 (2012)
46. Santhosh, B., Manjaiah, D.H.: An improved task scheduling algorithm based on max-min for cloud computing. International Journal of Innovative Research in Computer and Communication Engineering. **2**(2), 84–88 (2014)
47. Wang, G, et al.: Towards synthesizing realistic workload traces for studying the hadoop ecosystem. 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems. IEEE (2011)