IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 12, 2018, accepted October 9, 2018, date of publication October 15, 2018, date of current version November 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2876033

Heuristic Cloudlet Allocation Approach Based on Optimal Completion Time and Earliest Finish Time

JEAN PEPE BUANGA MAPETU[1], ZHEN CHEN[1,2], AND LINGFU KONG[1]

[1]School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China
[2]The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao 066004, China

Corresponding author: Zhen Chen (zhenchen@ysu.edu.cn)

This work was supported in part by the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grant 2017ZX05019001-011, in part by the National Natural Science Foundation of China under Grant 61772450, in part by the China Postdoctoral Science Foundation Grant under Grant 2018M631764, in part by the Hebei Postdoctoral Research Program under Grant B2018003009, and in part by the Doctoral Fund of Yanshan University under Grant BL18003.

ABSTRACT Cloud computing is an information technology paradigm that enables ubiquitous access to shared pools of configurable system resources and higher level services required by modern technology. Task scheduling is an important part in cloud computing for limited number of heterogeneous resources and increasing number of user tasks. Task scheduling is to allocate tasks (cloudlets) to the best suitable resources to increase performance in terms of some parameters, such as makespan and resource utilization. Allocating cloudlets with good load balancing and minimum makespan is an NP-hard optimization problem. Many meta-heuristic and heuristic algorithms have been proposed to solve the said problem, but they lack in considering the completion time of virtual machine and total length of its allocated cloudlets instead of only considering completion time of a cloudlet. This lack leads to decrease the performance of a cloud system in some cases, such as large cloudlets. To address the said problem, in this paper, we propose an optimal heuristic cloudlet allocation algorithm for resource allocation and task scheduling, referred as HCA, to cope with the increasing large number of user cloudlets under minimum resource capacity. So, we devise a new mechanism to combine optimal completion time and earliest finish time to minimize both degree of imbalance and overall completion time. The experimental results show that the proposed HCA can achieve effectively and efficiently good performance, best load balancing, and improve the resource utilization in comparison with the other existing cloudlet allocation methods.

INDEX TERMS Cloud computing, cloudlet allocation, optimal completion time, earliest finish time, load balancing.

I. INTRODUCTION

In recent years, cloud computing has emerged as heterogeneous distributed computing system to manage and allocate computing resources to user applications over the Internet in a self-service, dynamically scalable and metered manner [1], [2]. User application can be divided in small size or high size of tasks (cloudlets). The advantage of high size is that the number of cloudlets is more reduced than with small size. However, a great number of large cloudlets whose cloudlet corresponds to high size lead the cloud system to fail in good load balancing and lower completion time. This is caused by their high execution time and delay during their execution on virtual machine (VM). Due to the energy consumption constraint, the number of virtual machines is limited to the capacity of physical machines [3]–[6]. Further, due to the security issues related to virtualization [7], all these VMs are distributed among increasing number of cloud users to complete running of their cloudlets. This means that the number of these VMs is much more reduced per single user. Thus, optimizing task scheduling with good load balancing and minimum makespan which takes into account of large and small cloudlets under resources capacity constraints, such as number of VMs and processing speed, still remains the major challenging issue not well solved in cloud computing. Therefore, task scheduling being an NP-hard problem, then a best optimal heuristic cloudlet allocation is required for

managing high number of large and small cloudlets submitted by a single user to maximize both load balancing and resource utilization and to minimize overall completion time.

Currently, many algorithms for resource allocation and task scheduling in cloud computing, have been proposed to improve load balancing, to minimize makespan, waiting time and to maximize resource utilization. These proposed algorithms can be divided into two categories that are as follows: heuristic algorithms [8]–[13], such as Min-Min, Max-Min [11] and meta-heuristic algorithms [14]–[18]. Further, in case of scheduling both large cloudlets and small cloudlets, those of both algorithms failed in lowest degree of load imbalance and lowest completion time of a resource at same time. With the increasing number of large cloudlets, recent heuristic algorithms [12], [13] suffer from imbalance load among VMs or hosts which highly increases makespan and leads the cloud system to the under-load and overload situations.

In order to improve task scheduling, many techniques based heuristic algorithms have been implemented such as heterogeneous earliest finish time (HEFT) [10] and minimum completion time (MCT) [11]. MCT assigns cloudlets in random order to the VM having earliest completion time. In this method, some tasks are allocated to the VMs without having minimum execution time. HEFT sorts firstly a list of cloudlets in decreasing order of upward rank calculated on basis of execution time and then, assigns a cloudlet to VM with earliest finish time using insertion-based approach. In our work, these two heuristic techniques are modified and combined in order to achieve a good task scheduling with best load balancing and minimum makespan.

Due to the increasing of large amount users which involves a large amount number of cloudlets, researchers have put many efforts on improving load balancing among VMs. Authors in [19] presented task scheduling algorithm based on particle swarm optimization which distributes the load among the virtual machine in order to minimize the overall response time. Authors in [20] proposed a heuristic approach which combines modified analytic hierarchy process, bandwidth aware divisible scheduling, and BAR optimization, longest expected processing time preemption (LEPT) and divide-and-conquer methods to maximize the utilization of computing resources under bandwidth and load on the virtual machine, as constraints. This approach uses LEPT and divide-and-conquer methods to check load of each VM and to balance it among all VMs in overloading case. Authors in [21] proposed load balancing algorithm based on honey bee behavior to distribute workload in the way that avoid underutilization and over-utilization of the resources. It allocates the incoming cloudlet to a VM on basis of number of cloudlets and processing time threshold value to achieve fairness and avoid congestion. However, this processing time threshold value can be higher, then leads the cloud system to fail in good load balancing. Authors in [12] devised range wise busy-checking 2-way balanced (RB2B) to achieve load balancing. RB2B focuses on distributing the number of cloudlets to the VMs in a uniform way with a balance threshold and earliest finish time. The RB2B algorithm considers the processing speed of each VM and cloudlet length range. Its experimental results showed its improvements on the heuristic algorithms which are as follows: min-min, max-min, resource aware scheduling algorithm, and conductance algorithm. However, these algorithms do not consider the total length of global queue of cloudlets and they lack in finding common completion time among heterogeneous VMs which may ensure good load balancing, in case of large and small cloudlets.

Based on task scheduling, many heuristic algorithms have been proposed to minimize the execution time of a task and finish time such as first strategy algorithm (FSA) [13], round robin (RR) [9] and standard deviation based modified cuckoo optimization algorithm (SDMCOA) [22]. FSA is based on a deadline, length of cloudlets and speed of execution of VMs, and vector that defines the number of cloudlets per VM, is distributed to each VM. FSA lacks in considering the completion time of VM and size of large cloudlets. As a result, this lack leads to higher completion time and improper load balancing. RR uses the ring as its queue to store cloudlets, and it allocates resources in circular order without using priority of the cloudlets. Each cloudlet in a queue has a same fixed unit of time, called quantum, allocated by scheduler and it will be executed in turn. If a cloudlet is unable to complete during its turn, it will be stored back to the queue waiting for the next turn. Large cloudlets are often assigned to the VMs with low MIPS (million instructions per second) increasing waiting time and overall completion time. As a meta-heuristic method, SDMCOA allocates cloudlets to suitable virtual machines by using fitness function based on finding maximum finish time. However, all these algorithms lack in finding a common completion time among heterogeneous VMs which may impact better on load balancing and overall completion time. In addition, there is no need to consider number of cloudlets, but the size of large and small cloudlets has to be considered. However, these heuristic algorithms do not focus on minimizing the completion time of each VM instead of only minimizing the execution time of a cloudlet which can increase the overall completion time.

In this paper, we mainly focus on designing an optimal heuristic cloudlet allocation algorithm for resource allocation and task scheduling, referred as HCA to cope with the increasing large amount number of cloudlets submitted by a single user over heterogeneous virtual machines. The proposed algorithm is divided in two phases, namely allocation based on optimal completion time and allocation based on earliest finish time. We devise a new mechanism to combine optimal completion time and earliest finish time to minimize both degree of imbalance and overall completion time and to maximize resource utilization. In allocation based on optimal completion time, the optimal completion time is calculated as common minimum completion time for all VMs leased by provider to ensure proper load balancing between all VMs and to impact on overall completion time.

From global queue, user cloudlets are assigning to the VMs under this completion time. In allocation based on earliest finish time, the unallocated user cloudlets are assigning to the VMs under earliest finish time in order to minimize waiting time, completion time of cloudlets and to maximize resource utilization. We simulate the proposed algorithm and the recent heuristic algorithms, such as RB2B, FSA and RR. To measure the performance of the proposed algorithm, we compare the algorithm against the recent ones using various metrics such as degree of imbalance (*DI*), makespan, resource utilization (*RU*), running time (*RT*) and waiting time of a cloudlet (*WT*).

Our main research contributions in this paper are as follows:

(1) A heuristic load balancing and task scheduling method is designed to minimize completion time, as makespan and to maximize utilization of heterogeneous virtual machines under optimal completion time and earliest finish time.

(2) We propose optimal completion time to ensure proper load balancing and to impact on overall completion time as well as makespan.

(3) We devise a new strategy to combine optimal completion time and earliest finish time to minimize both degree of imbalance and overall completion time. This strategy avoids overloaded and under-loaded cases. Comparing with recent heuristic and meta-heuristic algorithms which are complex methods, our strategy is simple in a way that is required for a good scheduler to achieve good performance of cloud system [23].

(4) We performed extensive experiments under different application scenarios, and successfully confirmed that our proposed approach is feasible for improving both load balancing and the resource utilization in comparison with the other existing counterparts. This is in case of any number of VMs, independent small cloudlets, independent large cloudlets and real workloads.

The remainder of this paper is organized as follows: Section II states the main problem to be resolved of this paper; Section III gives its solution through a proposed optimal heuristic algorithm for task scheduling and resource allocation, impact analysis of proposed solutions and its complexity analysis; Section IV shows experimental results and analysis and Section V draws conclusions.

II. BALANCE RULE ANALYSIS AND PROBLEM STATEMENT

A. BALANCE RULE ANALYSIS

Definition 1 (Virtual Machine): A virtual machine (VM) can be described as a tuple VM=$\{id, mips, bw, pesnumber\}$, where *id* represents identifier of a *VM*, *mips* represents the processing speed per processing element (PE) at a *VM*, *bw* represents bandwidth of a *VM* and *pesnumber* represents the number of PE in a *VM*.

Definition 2 (Cloudlet): A cloudlet (C), referred to a task in this paper, can be described as tuple $C = \{id, length, pesnumber\}$, where *id* represents identifier of a C, length represents the size of C in MI (million instructions) and *pesnumber* represents the number of PE for running a C on a suitable *VM*.

Definition 3 (Small Cloudlet): A small cloudlet can be defined as a cloudlet whose length is small and needs a short execution time [11].

Definition 4 (Large Cloudlet): A large cloudlet can be defined as cloudlet whose length is large and needs a long execution time. Based on work [24], a large cloudlet can be viewed as a set of dependent cloudlets with no communication cost. So, in this paper, it will not be decomposed in multiple sequences of cloudlets to be executed on multiple virtual machines in order to minimize the communication cost at value of zero.

Assuming that there are m *VM*s, and n independent cloudlets (small or large cloudlets), let $et_{i,j}$ be the execution time for cloudlet C_j corresponding to VM_i, as shown by (1).

$$et_{ij} = \frac{C_j\ .length}{VM_i\ .pesnumber \times VM_i\ .mips} \tag{1}$$

where C_j *.length* denotes size of cloudlet C_j in MI, VM_i *.pesnumber* denotes number of PE in a VM_i, and VM_i *.mips* denotes execution speed of VM_i in MIPS. Thus, m *VM*s and n cloudlets will construct $m \times n$ cloudlet allocation matrix CA.

$$\mathrm{CA} = \begin{bmatrix} et_{1,1} & et_{1,2} & et_{1,3} & \dots & \dots & \dots & et_{1,n-1} & et_{1,n} \\ et_{2,1} & et_{2,2} & et_{2,3} & \dots & \dots & \dots & et_{2,n-1} & et_{2,n} \\ \vdots & \vdots & \vdots & \dots & \dots & \dots & \vdots & \vdots \\ et_{m,1} & et_{m,2} & et_{m,3} & \dots & \dots & \dots & et_{m,n-1} & et_{m,n} \end{bmatrix}_{m \times n} \tag{2}$$

In the above matrix CA, each row represents the execution time of different cloudlets processed on a targeted VM, and each column represents the execution time of a cloudlet on different VMs. The sum of execution time $et_{i,j}$ on each row i, defines the completion time in VM_i.

Let CT_i denote completion time in targeted VM_i; it is defined by (3).

$$CT_i = \sum_{j=1}^{n} et_{i,j} \quad for\ i = 1, .., m \tag{3}$$

From the above formula and cloudlet allocation matrix, we deduce three rules for a balanced cloud system that are as follows:

Rule 1: Given a list of cloudlets. If all the allocated VMs have the same completion time, expressed by $CT_i = CT_{i+1}$, then degree of load imbalance is equal to zero, resource utilization equal to 100%. Hence, the cloud system is ideal and well balanced (**R1**).

Rule 2: If all the allocated VMs have closest completion time, denoted by $CT_i \cong CT_{i+1}$, then degree of load imbalance gets close to zero, resource utilization close to 100%. Hence, the cloud system is much more balanced. (**R2**).

Rule 3: If **R1** and **R2** are not satisfied, denoted by $CT_i \neq CT_{i+1}$, then the system is worse (**R3**).

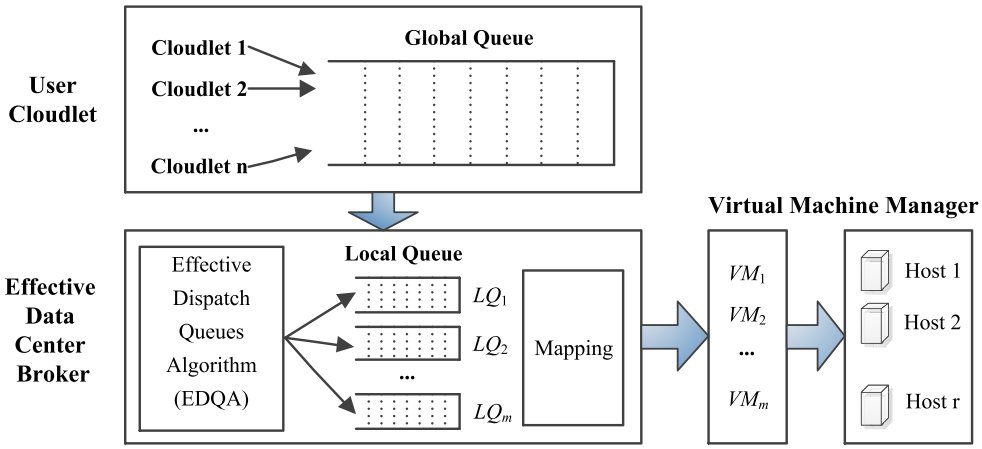


FIGURE 1. Heuristic cloudlet allocation framework.

Although rule **R1** is the ideal view to make a perfect balanced cloud system, it is much more complicated to achieve it because of heterogeneity of cloudlets and resource capacity constraints. While rule **R2** can be achieved, despite it is more difficult. So, this work focuses on rule **R2** to minimize load imbalance with purpose of maximizing resource utilization.

B. PROBLME STATEMENT

The problem under investigation can be described as follows. There are n independent cloudlets (small or large cloudlets) given by a user to be executed on m heterogeneous VMs. Let$CVM=\{VM_1, VM_2,\ldots,VM_m\}$ denote the list of virtual machines, and $CC=\{C_1, C_2, \ldots, C_n\}$ denote the list of independent cloudlets. In practical scenario, the number of cloudlet is larger than the number of *VM*s. Thus, the division of n/m will generally be greater than 1 which means more cloudlets than virtual machines, so that some *VM*s will need to be assigned multiple cloudlets. However, the execution time of each cloudlet is increased proportionally to its length. In addition, the completion time on each VM not only depends on execution time of each running cloudlet, but also on the total length of all the running cloudlets in that VM. So, as consequence, a great number of large cloudlets may increase highly overall completion time, decrease the efficiency of resource utilization and lead cloud system to fail in load balancing.

Considering the said consequence and referring to the balance rules, our problem can be formulated into three points, as follows: Minimizing load imbalance between highest loaded VM and lowest lighted VM, minimizing makespan and maximizing resource utilization under optimal completion time and earliest finish time.

III. OUR APPROACH

This section presents a description of our proposed framework, its corresponding algorithm and its pseudo-code (Algorithm 1) and defines the flowchart for the proposed approach.

A. FRAMEWORK

In our proposed architecture, virtual machines are created and allocated to the host(s) and are arranged in increasing order of processing speed. A cloudlet arrives from the global queue (GQ) to the effective datacenter broker (EDCB). EDCB is a cloud datacenter broker where the proposed HCA algorithm is implemented. Fig. 1 illustrates an overview of HCA framework. The framework consists of three modules, as follows:

The first module is the user cloudlet which represents the set of the cloudlets.

The second module is effective data center broker which outputs a cloudlet local queue for each VM. In EDCB, there are two sub-modules which are as follows: effective dispatch queues algorithm (EDQA) and mapping. EDQA computes and finds all cloudlets for each VM according to optimal completion time and earliest finish time constraints, while Mapping assigns each local queue to the corresponding VM.

The third module is virtual machine manager relying on different hosts. It manages a set of virtual machines that are used to execute user cloudlets. The cloudlets have been stored in an $m \times n$ matrix, where m is the number of virtual machines and n is the number of cloudlets.

B. OPTIMAL COMPLETION TIME

In order to maximize load balancing which implies to have closest completion time between all VMs, this paper defines the calculation of optimal completion time for all cloudlets given by a user, denoted as OCT at first phase of allocation of cloudlets to VMs.

Definition 5: Optimal completion time (OCT) is defined as common minimum completion time distributing to all VMs in order to queue a set of cloudlets; it is given by (4).

$$OCT = min\{CT_i/m\}_{i=1\cdots m} \tag{4}$$

where CT_i represents completion time in $VM_i (i = 1 \ldots m)$ following (3), and m, number of VMs. It is oriented virtual machine.

Here, we give the demonstration of eq. (4).

Consider a collection of n cloudlets $CC = \{C_1, C_2, \cdots, C_n\}$ and m virtual machines, $CVM = \{VM_1, VM_2, \cdots, VM_m\}$.

Let TLC, TLQ, CT_i, $CTLQ_i$ denote total length of CC, total length of a subset of CC, completion time in VM_i for TLC, completion time in VM_i for TLQ.

According to (1) and (3), we derive (5).

$$\begin{aligned} CT_i = & \frac{C_1 \ .length}{VM_i \ .pesnumber \times VM_i \ .mips} \\ & + \frac{C_2 \ .length}{VM_i \ .pesnumber \times VM_i \ .mips} \\ & + \cdots \\ & + \frac{C_n \ .length}{VM_i \ .pesnumber \times VM_i \ .mips} \end{aligned}$$

Then,

$$CT_i = \frac{C_1 \ .length + C_2 \ .length + \cdots + C_n \ .length}{VM_i \ .pesnumber \times VM_i \ .mips} \quad (5)$$

But,

$$TLC = C1 \ .length + C2 \ .length + \cdots + C_n \ .length \quad (6)$$

TLC corresponds to total length of global queue of all user cloudlets.

$$\text{So, (6) in (7)} \Rightarrow CT_i = \frac{TLC}{VM_i \ .pesnumber \times VM_i \ .mips} \quad (7)$$

By applying an equal distribution of TLC to all the m VMs, we deduce TLQ (8).

$$TLQ = \frac{TLC}{m} \quad (8)$$

TLQ corresponds to total length of user cloudlets executing by each VM.

Then, let us find completion time for each VM to process TLQ (9).

$$CTLQ_i = \frac{TLQ}{VM_i \ .pesnumber \times VM_i \ .mips} \quad (9)$$

$$\begin{aligned} (8) \ in \ (9) \Rightarrow CTLQ_i &= \frac{TLC}{m} \\ &\times \frac{1}{VM_i \ .pesnumber \times VM_i \ .mips} \\ \Rightarrow CTLQ_i &= \frac{1}{m} \times \left(\frac{TLC}{VM_i \ .pesnumber \times VM_i \ .mips} \right) \end{aligned} \quad (10)$$

$$S = \frac{TLC}{VM_i \ .pesnumber \times VM_i \ .mips} \quad (11)$$

$(11) = (7) = (3) \Rightarrow CTLQ_i = \frac{CT_i}{m}$

Considering definition 3,

$$OCT = min\,(CTLQ_i) \Leftrightarrow OCT = min\left(\frac{CT_i}{m}\right)$$

C. EARLIEST FINISH TIME

In order to reduce the execution time of each unallocated cloudlet, we define earliest finish time, denoted as EFT, at second phase of allocation of cloudlets.

Definition 6: Earliest start time (EST) is the earliest possible time that a cloudlet begins after the time that all predecessors are supposed to complete their execution. In our case, EST is initialized to CT determined at the first phase of the proposed algorithm, as shown by (12).

$$EST_i = CT_i \quad (12)$$

Definition 7: Earliest finish time (EFT) is the earliest possible time that cloudlet can be finished from EST, as shown by (13). It is oriented cloudlet.

$$EFT_{i,j} = et_{i,j} + EST_i \quad (13)$$

where EST_i is the earliest execution start time for cloudlet C_j on VM_i.

D. IMPACT ANALYSIS OF OPTIMAL COMPLETION TIME AND EARLIEST FINISH TIME

This section gives the impact of optimal completion time (OCT) and earliest finish time (EFT) through the analysis of these two following scenarios:

(1) Among all VMs, some VMs are running high number of large cloudlets than small cloudlets.

(2) Among all VMs, some VMs are running high number of small cloudlets than large cloudlets.

The first scenario may be similar to Max-Min algorithm which prioritizes running large cloudlets than small cloudlets [11]. Due to the high number of large cloudlets, VM may first run large cloudlets, then small cloudlets. Based on works [11], [25], this scenario may lead the cloud system to high makespan, to high degree of load imbalance, and then to poor resource utilization when large cloudlets outnumber small ones.

The second scenario may be similar to Min-Min algorithm which prioritizes running small cloudlets than large cloudlets [11]. Due to the high number of small cloudlets, VM may first run small cloudlets, then large cloudlets. Referring to the work [12], this scenario may lead the cloud system to high makespan, to high degree of load imbalance, and then to poor resource utilization when the number of small cloudlets is too much.

In order to avoid these two scenarios which lead the cloud system to fail in proper load balancing, this paper proposed the distribution of cloudlets under combination of OCT and EFT. The sequence order of this distribution is presented as follows. At the first phase, we consider the allocation on completion time to maintain a proper load balancing. The algorithm firstly calculates OCT, as shown by (4) in section III (B). Unlike to max-min and min-min, each VM receives OCT, then large and small cloudlets are assigned to that VM with respect to OCT by using FCFS method (First come first serve). This is in order to maintain the same completion

time among all VMs. If OCT is maximized, then maximum completion time can be equal to OCT. Hence, the overall makespan will be highly increased. So, OCT must be minimized. As results, low overall makespan, high load balancing level and high resource utilization rate at this phase. After completing the first phase, the second phase based on allocation under EFT is activated to distribute all unallocated cloudlets among all VMs. For each unallocated cloudlet, EFT is firstly calculated, as shown by (13) in section III (C), then it is allocated to a VM which presents a minimum EFT. Hence, this phase will maintain a low overall makespan and lead cloud system to high load balancing level and high resource utilization.

E. HEURISTIC CLOUDLET ALLOCATION ALGORITHM

Our proposed HCA algorithm is described, in two phases, as follows: allocation based completion time phase and allocation based earliest finish time phase. At allocation based completion time phase, optimal completion time based on total length of set of user cloudlets, resource capacity and number of heterogeneous virtual machines leased by a provider is computed; and then, a set of cloudlets which the estimated completion time is not greater than *OCT*(optimal completion time), are allocated to a virtual machine (line 3-4). At allocation based earliest finish time phase, earliest finish time based on execution time of each unallocated cloudlet and its earliest start time on a virtual machine is computed. The cloudlet with lowest *EFT* (earliest finish time) corresponding to a virtual machine is allocated to that virtual machine (line 5-6).

Algorithm 1 Heuristic Cloudlet Allocation (HCA)

Input: A global queue (*GQ*) of n cloudlets (C) and list of m virtual machines (*VM*)
Output: Allocation of all cloudlets on suitable VMs
First phase: Completion time
1. Arrange a set of virtual machines increasing order of processing speed
2. Submit a list of cloudlets to EDCB
3. Calculate OCT as shown by (4)
4. EDQA targets a VM, and selects all cloudlets whose completion time in the targeted VM is less than OCT. Then, remove all allocated cloudlets from GQ.
Second phase: Earliest finish time
5. EDQA checks whether all cloudlets from global queue are queued to be mapped on VMs, after completion of the first phase. If the condition is satisfied, then terminate the second phase.
6. EDQA assigns the non-allocated cloudlets to VMs according to EFT, as shown by (13), until GQ become empty.

F. FLOWCHART FOR HCA ALGORITHM

Figure 2 shows a flowchart of HCA Algorithm. This flowchart defines three main parts, which are as follows: computation of optimal completion time, allocation under optimal completion time and allocation under earliest finish time. The principal difference between the second part and the third part is that the second part focuses on minimizing completion time of a VM as well as completion time of its all assigned cloudlets whereas the third part focuses on minimizing execution time of each cloudlet.

G. COMPLEXITY ANALYSIS

In order to obtain the time complexity of the algorithm, we analyze the time complexity of its two phases respectively.

In allocation based on optimal completion time phase, the most complicated computations are calculating optimal completion time and allocating a set of cloudlets from global queue to each virtual machine (VM) under the calculated completion. Hence, the time complexity can be equal to $O(n)+O(m)+O(mn)$ where n and m are number of cloudlets and number of VMs, respectively. So, the time complexity of this phase is $O(mn)$.

In allocation based on earliest finish time phase, all unallocated cloudlets after the first phase are concerned in this phase. The main computations are as follows: calculating earliest completion time of a cloudlet on each virtual machine and allocating a cloudlet to suitable virtual machine with respect to the calculated time. The time complexity of this phase depends on the position of the last unallocated cloudlet in the global queue. In these cases, we define the worst-case and the best case running time. Let k be the position of a cloudlet in the global queue and it starts from the end of list by 0. In the worst case, $k = 0$; and then, the time complexity is $O(nm)$. In best case, $k > 0$; and, then, the time complexity is $O((n-k)m)$.

To sum up, the proposed HCA algorithm's time complexity is $O(mn)$. Finally, the time complexity is approximately equal to $O(n)$ in polynomial time since m is constant.

IV. EXPERIMENTS AND RESULTS

In this section, we conduct five groups of experiments to verify the performance of the proposed approach, which are dedicating to answer the following questions:

(1) How does HCA perform under short number of cloudlets with small size (small cloudlets)?

(2) How does HCA perform under great number of cloudlets with high size (large cloudlets)?

(3) How does HCA perform under real workloads?

(4) What is the impact of the number of virtual machines (VM)?

(5) What is the efficiency of HCA?

A. EXPERIMENT SETUP

We conduct experimental studies carried out by simulation, using CloudSim 3.0.3 simulator to evaluate the effectiveness of the proposed HCA algorithm under different application scenarios [26], [27]. All algorithms were written using Netbeans 8.2 Java programming language and running on a computer with Core i5-6500 CPU, 3.2GHz, 4G RAM. The experimental parameter settings are shown in Table 1.

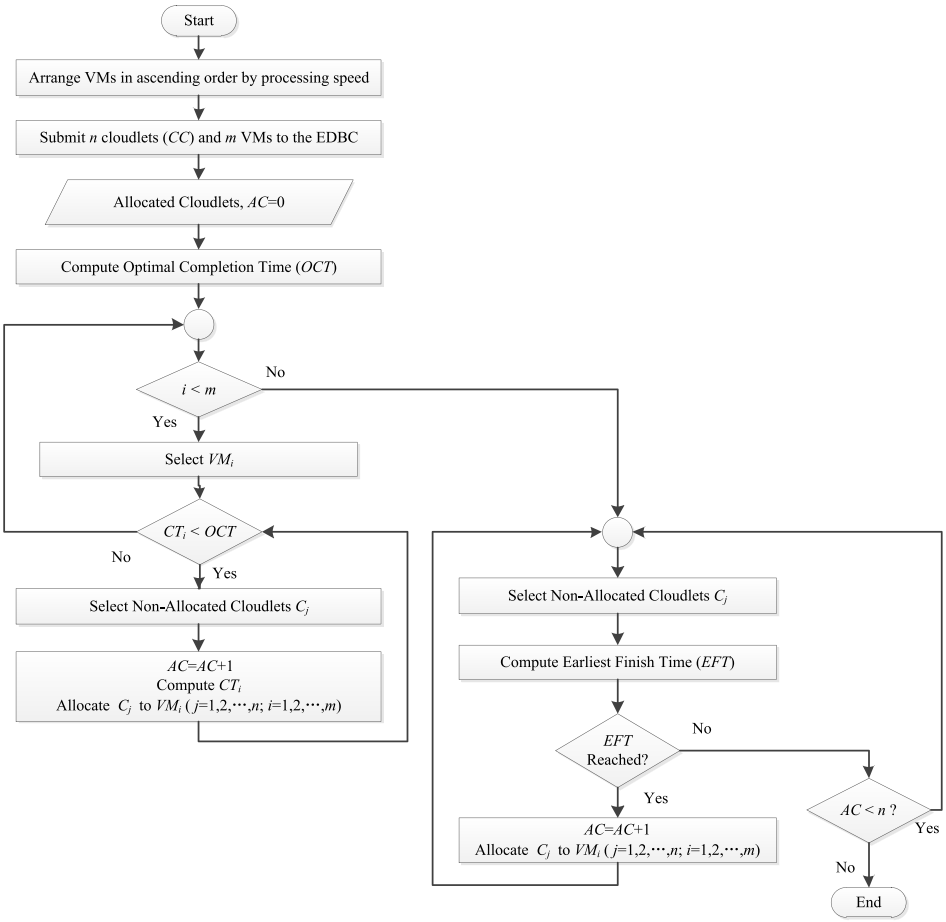


FIGURE 2. Flowchart for HCA Algorithm.

For evaluation purpose, we compare our HCA algorithm with three other methods, named RR (Round robin) [9], RB2B (Range wise busy checking 2 way balanced) [12], and FSA (First strategy algorithm) [13].

RR focuses on fairness and on distributing the list of cloudlets equally to all VMs. Each cloudlet in a queue has same execution time, called quantum and it will be executed in turn. This algorithm starts with a cloudlet and moves on to the next cloudlet, after a VM is assigned to that cloudlet until all VMs have been allocated at least one VM. Then, it returns to the first cloudlet again. As an example, if there are four cloudlets and four VMs, each cloudlet would be allocated one VM.

RB2B focuses on distributing a batch of cloudlets to the VMs in a most uniform way under cloudlet length acceptance range, balance threshold, local queue length limitation and earliest finish time, as constraints to achieve load balancing. At first, this algorithm sorts a list of VMs by processing speed (MIPS) in ascending order; then, it defines three phases to distribute a list of cloudlets from a user. In first phase, this algorithm defines a cloudlet length acceptance range for each VM. Then, it measures the length of each cloudlet, and targets a VM following cloudlet length acceptance range. In the second phase, it checks both the availability of targeted VM and balance threshold condition to allocate the cloudlet to that VM. In the third phase, the algorithm searches for a VM according to earliest finish time (EFT), balance threshold and local queue (LQ) length limitation to allocate a cloudlet.

FSA is based on assigning cloudlets to suitable VMs under vector of number of cloudlets, as balance threshold, deadline of a cloudlet, cloudlet length and processing speed of VM. This FSA algorithm has three steps. First, it sorts a list of cloudlets by instruction deadline and length in ascending order; second, it sorts a list of VMs by processing speed (MIPS) in ascending order; third, FSA calculates the vector of number of cloudlets for each VM. Then, it applies FCFS method to distribute a list of cloudlets among a list of VMs with respect to calculated vector.

B. METRICS

The performance metrics in our experiments are overall degree of imbalance (DI), average resource utilization (ARU), makespan and average waiting time of cloudlet (AWT). *DI* is calculated by (14).

$$DI = (T_{max} - T_{min})/T_{avg} \tag{14}$$

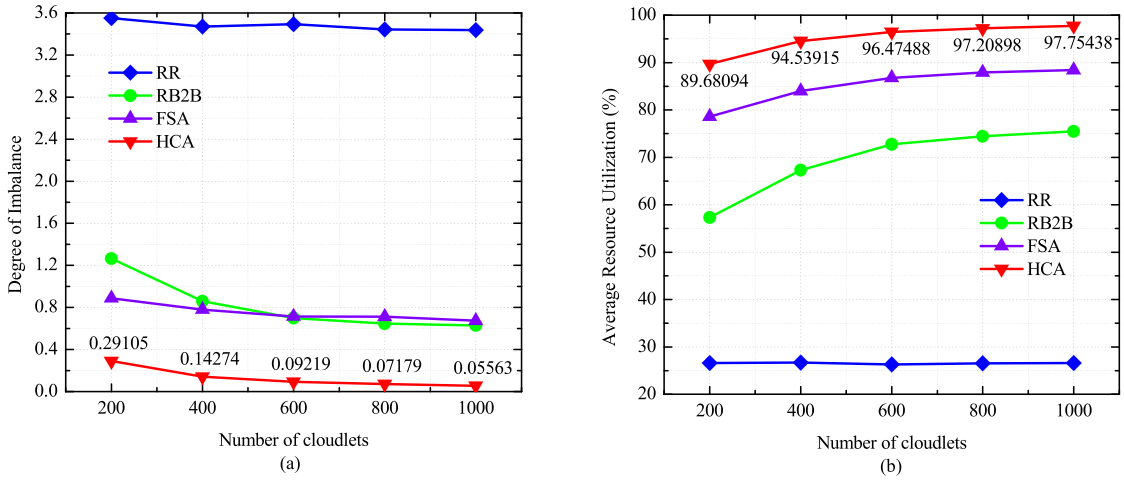


FIGURE 3. Comparisons with short number of cloudlets with small size on DI and ARU (small cloudlets). (a) DI (b) ARU.

TABLE 1. Experimental parameter settings.

Type	Parameter	Value
Datacenter	Number of Datacenter	1
	Number of Host	6/200
	Type of Manager	Time Shared
	Number of PE per Host	4
	Bandwidth	10000/100,000
	Operating system	Linux
	Hypervisor	Xen
	Host Memory (MB)	10240/102,400
Virtual Machine (VM)	Total number of VMs	5 - 30/2,000
	Processing speed (MIPS)	500-10000/5,000-20,000
	Number of PE per VM	1/5
	VM Memory (MB)	512
	Bandwidth (Bit)	1024
	Type of Manager	Space Shared
Cloudlet	Length of Cloudlet (MI)	1000 – 2,000,000
	Number of PE per requirement	1

where T_{max}, T_{min} and T_{avg} are execution time maximum, execution time minimum and average total execution time of all VMs respectively.

ARU is gaining significance as service providers want to earn maximum profit by renting limited number of resources. *ARU* is computed by (15).

$$ARU = \frac{\sum_{i=1}^{m} CT_i}{makespan \times m} \tag{15}$$

where m is the number of allocated VMs and CT_i, completion time described by (3).

Makespan is the maximum completion time of all VMs for running user cloudlets which is expressed as (16).

$$makespan = \max_{1 \le i \le m} \{CT_i\} \tag{16}$$

Note that small values of makespan mean that the scheduler is providing good and efficient planning of tasks to resources.

C. PERFORMANCE WITH SHORT NUMBER OF CLOUDLET WITH SMALL SIZE

To study the performance of our proposed HCA algorithm without real workload for small cloudlets, the size of cloudlet is set between 1000 MI and 100,000 MI in random distribution executed on 20 heterogeneous VMs which are arranged in uniform distribution by increasing order and distributed among 6 hosts. The processing speed of VM is set between 500 MIPS and 10,000 MIPS.

In comparison with FSA, RB2B and RR, Fig. 3(a) shows that degree of imbalance in the proposed HCA algorithm gets closest to zero when the number of small cloudlets increases. First, this is because many cloudlets are allocated to each VM with high value of OCT (optimal completion time) which depends on total length of all cloudlets, as described in (4) and (5). Second, it is because of equal distribution of these cloudlets with respect to OCT. As shown in the formula of degree of imbalance (14) and considering the results, we can deduce that most of all VMs have closest completion time. Hence, HCA achieves a best load balancing for small cloudlets.

Fig. 3(b) shows that our HCA outperforms FSA, RB2B and RR in term of resource utilization. For instance, with 1000 small cloudlets, ARU with HCA is closest to 100%. Regarding these results, we can say that HCA keeps resources more busy than other algorithms.

In comparison with FSA, RB2B and RR, Table 2 presents the results and minimum improvement of the proposed HCA algorithm in term of makespan and waiting time. First, the average waiting time of a cloudlet produced by HCA

TABLE 2. Comparisons with short number of cloudlets with small size on makespan and AWT.

Metric	Makespan(unit:s)					AWT(unit:s)				
Number of cloudlets	200	400	600	800	1000	200	400	600	800	1000
RR	530	1026	1557	2052	2556	72	139	207	275	342
RB2B	188	298	402	522	639	52	97	144	192	238
FSA	*108*	*201*	*289*	*381*	*473*	*47*	*90*	*132*	*174*	*218*
HCA	**95**	**185**	**275**	**364**	**454**	**50**	**96**	**142**	**187**	**232**
Improvement	12.03%	7.96%	4.84%	4.46%	4.01%	-6.38%	-6.66%	-7.57%	-7.47%	-6.42%

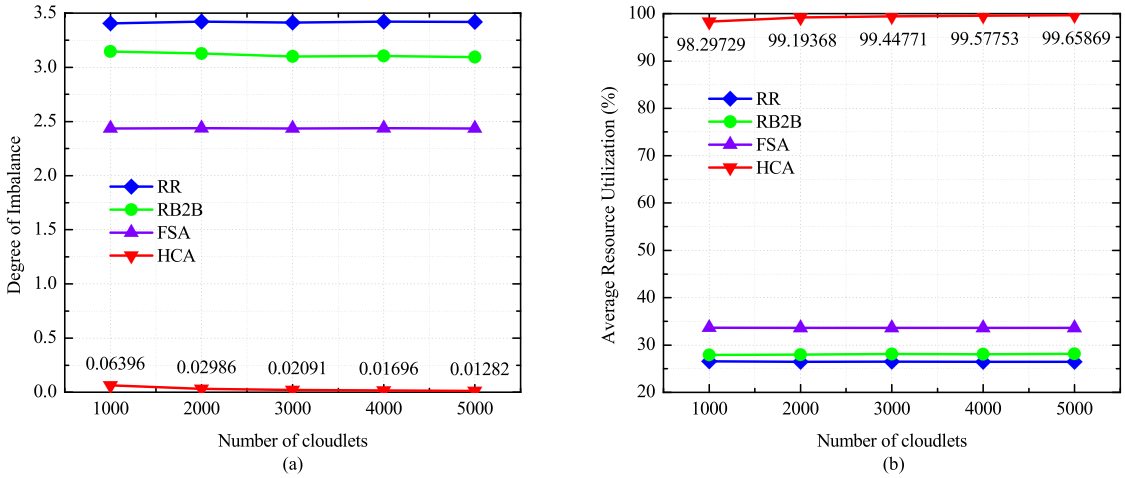


FIGURE 4. Comparisons with great number of cloudlets with high size on DI and ARU (large cloudlets). (a) DI. (b) ARU.

is little less than RB2B and much less than RR. However, compared to FSA, it is a little high. The minimum rate of improvement is negative which means no improvement in comparison with FSA, one of existing algorithms. This is because FSA uses equal distribution of cloudlets based on number of cloudlets and sorts a list of cloudlets which reduce delay between two cloudlets. Second, we observe that HCA produces the smallest makespan which is close to FSA and much smaller than RB2B and RR. Due to the equal distribution of number of cloudlets, low VM (VM with the smallest processing speed) is as same as high VM (VM with the highest processing speed) which leads to high makespan. However, waiting time in FSA is less than in HCA. That is why FSA produces makespan closer to HCA.

Despite the results of waiting time, the proposed HCA algorithm is suitable to task scheduling and load balancing for small cloudlets. So, a cloud user can get more satisfaction to complete running of all its cloudlets in short time.

D. PERFORMANCE WITH GREAT NUMBER OF CLOUDLET WITH HIGH SIZE

Based on definition 4 in section II (A), a large cloudlet is regarded as a set of dependent cloudlets with no communication cost [28] in this section. To study the performance impact of our proposed algorithm without real workload for large cloudlets, we set cloudlet size between 1,000,000 MI and 2,000,000 MI randomly; cloudlets are executed on 20 heterogeneous *VM*s which are arranged in uniform distribution by increasing order and distributed among 6 hosts. The processing speed of VM is set between 500 MIPS and 10,000 MIPS.

Fig. 4(a) shows that HCA produces far better minimum degree of imbalance than FSA, RB2B and RR. This is because of distribution of cloudlets among VMs with respect to OCT and EFT (earliest finish time) without determining in advance a balance threshold or prioritizing a cloudlet. These results demonstrate that HCA has the best effect on load balancing to manage large cloudlets under virtual machines capacity constraints.

Fig. 4(b) shows that the resource utilization is far better maximized with HCA than FSA, RB2B and RR due to achievement of load balancing based on OCT and EFT. We can deduce that HCA produces closest completion time between all virtual machines. Hence, HCA provides the highest utilization of heterogeneous resources.

The Table 3 shows the results of HCA in comparison with FSA, RB2B and RR in term of makespan and waiting time. Unlike to the experiment in section IV (C) in term of waiting time, the results in Table 3 shows a high minimum rate of

TABLE 3. Comparisons with great number of cloudlets with high size on makespan and AWT.

Metric	Makespan(unit:s)					AWT(unit:s)				
Number of cloudlets	1000	2000	3000	4000	5000	1000	2000	3000	4000	5000
RR	74735	149963	224708	300121	374956	10032	19960	29899	39794	49709
RB2B	68704	136492	202886	270854	337605	9398	18514	27589	36663	45775
FSA	*51262*	*102590*	*153899*	*205238*	*256383*	*8768*	*17389*	*26028*	*34647*	*43256*
HCA	**13191**	**26268**	**39271**	**52340**	**65388**	**6638**	**13180**	**19684**	**26217**	**32753**
Improvement	74.26%	74.39%	74.45%	74.49%	74.49%	24.29%	24.20%	24.37%	24.33%	24.28%

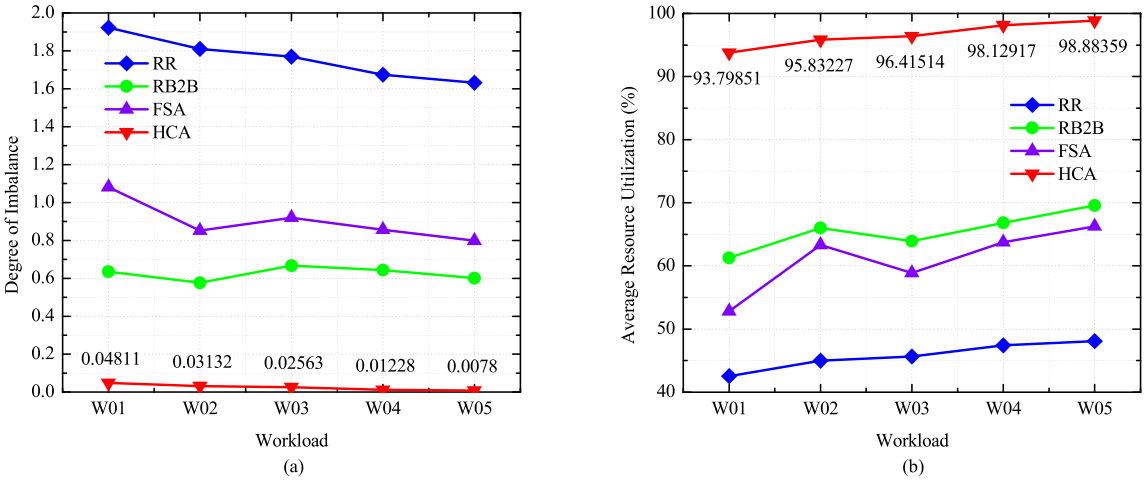


FIGURE 5. Comparisons with different real workload on DI and ARU. (a) DI. (b) ARU.

improvement with HCA. This is because HCA focuses on distributing cloudlets with respect to OCT and EFT which range the cloudlets in special execution order to minimize delay. This shows the inefficiency of FSA to minimize delay between cloudlets by sorting them. Considering the results of makespan, we observe that the rate of improvement in HCA is much high than other existing algorithms, as RR, RB2B and FSA. Unlike to RR, RB2B and FSA, this is because our proposed HCA algorithm does not only focus on minimizing execution time, but on minimizing completion time.

Regarding these results, it can be concluded that HCA achieves task scheduling with best load balancing and minimum makespan under resource capacity constraints for large cloudlets. Further, our proposed HCA algorithm can deal with dependent cloudlets in minimizing the communication cost accordingly with definition of large cloudlet.

E. PERFORMANCE WITH REAL WORKLOADS

In this section, we conducted the experiment on five different real workloads with 2,000 heterogeneous virtual machines (VMs) to check the performances of the heuristic algorithms for task scheduling and load balancing for IaaS cloud computing. We considered four different types of VMs with, 5,000 MIPS, 10,000 MIPS, 15,000 MIPS and 20,000 MIPS.

TABLE 4. Description of real workloads.

Workload	Name	Number of cloudlet	Desccription
W01	LLNL-Thunder	128,662	large Linux cluster called Thunder installed at Lawrence Livermore National Lab
W02	LCG	188,041	Large Hadron Collider Computing Grid in London
W03	DAS2-fs0	225,711	Distributed ASCI Supercomputer-2 in the Netherlands
W04	RICC	447,794	RIKEN Integrated Cluster of Clusters in Japan
W05	PIK-IPLEX	742,965	IBM iDataPlex cluster at the Potsdam Institute for Climate Impact Research (PIK) in Germany

Theses VMs are distributed among 200 hosts. These workloads are used to evaluate the performance by degree of imbalance, resource utilization, makespan and waiting time

TABLE 5. Comparisons with different real workloads on makespan and AWT.

Metric	Makespan(unit:s)					AWT(unit:s)				
Workload	W01	W02	W03	W04	W05	W01	W02	W03	W04	W05
RR	7039	10138	11937	22498	36537	1515	2298	2743	5352	8796
RB2B	*3973*	*5597*	*6978*	*13057*	*20658*	1204	1815	2179	4239	6964
FSA	4443	5618	7388	13292	21065	*1227*	*1849*	*2211*	*4288*	*7045*
HCA	**2405**	**3613**	**4293**	**8293**	**13602**	**1172**	**1778**	**2115**	**4116**	**6771**
Improvement	39.46%	35.44%	38.47%	36.48%	34.15%	2.65%	2.03%	2.93%	2.90%	2.77%

TABLE 6. Comparisons with different number of virtual machines on makespan and AWT.

Metric	Makespan(unit:s)					AWT(unit:s)				
Number of VMs	5	10	15	20	30	5	10	15	20	30
RR	3575	3251	2875	2598	2129	868	579	430	343	246
RB2B	1678	1068	792	640	451	693	424	302	237	163
FSA	*1452*	*865*	*616*	*472*	*340*	*631*	*389*	*279*	*217*	*156*
HCA	**1344**	**818**	**581**	**454**	**313**	**679**	**417**	**297**	**231**	**161**
Improvement	7.44%	5.43%	5.68%	3.81%	7.94%	-7.60%	-7.19%	-6.45%	-6.45%	-3.20%

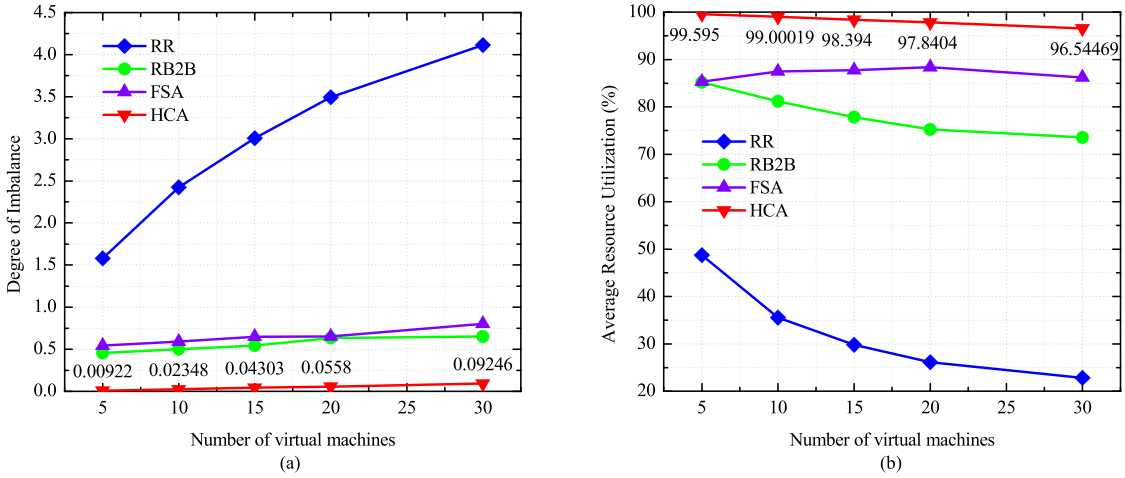


FIGURE 6. Comparisons with different number of virtual machines on DI and ARU. (a) DI. (b) ARU.

in heterogeneous environment. They are presented as W01, W02, W03, W04 and W05 respectively which are generated from Parallel Workload Archives [29]. The description of real workloads is given by Table 4.

Regarding the results of degree of imbalance in Fig. 5(a), HCA minimizes degree of imbalance much less than FSA, RB2B and RR. For instance, with W04, HCA produces DI = 0.012, while DI = 0.856 with FSA, DI = 0.644 with RB2B and DI = 1.674 with RR. This is because HCA determines a common minimum completion time which controls the load balance in each VM in order to avoid over-loaded VMs and under-loaded VMs. These results mean that our proposed HCA algorithm achieves a best load balancing. In addition, they mean that the cloud system is available to receive new incoming workloads.

Fig. 5(b) shows that the average resource utilization is far better maximized with HCA than FSA, RB2B and RR. For instance, with W05, the results in HCA are closest to 100% due to the lowest degree of imbalance which means that no single VM among all VMs is underutilized. With low degree of imbalance, the completion time difference among all VMs is much more minimized. This means that

resource utilization is maximized according to formula (15). Therefore, the results in Fig. 5(b) show that the proposed HCA algorithm can provide the optimum usage of available heterogeneous resources.

In Table 5, we observe that the proposed HCA algorithm produces the best minimum makespan and waiting time is minimized in comparison with FSA, RB2B and RR. This is because HCA focuses on minimizing completion time and using a common completion time among all VMs leased by a cloud provider. The results in HCA mean that HCA can improve well the QoS (Quality of service) in cloud system such as availability and performance.

Comparing with the two above experiments, we observe that RB2B algorithm outperforms FSA algorithm in term of degree of imbalance, resource utilization, makespan and waiting time while HCA algorithm still outperforms all of them. Hence, our proposed algorithm is stable and effective.

We concluded that the proposed HCA algorithm can provide proper load balancing and task scheduling in case of real datasets.

F. IMPACT OF THE NUMBER OF VIRTUAL MACHINES

To investigate the impact of the number of VMs without real workload, we set a fixed number of cloudlets to 1000, and the number of VMs has been conducted with 5 VMs, 10 VMs, 15 VMs, 20 VMs and 30 VMs. The processing speed of VM is set between 500 MIPS and 10,000 MIPS. As compared with the small cloudlet, the size of cloudlet is set between 1000 MI and 100,000 MI in random distribution.

Fig. 6(a) shows that HCA is far better than FSA, RB2B and RR algorithms with the varying number of VMs from 5 to 30. For instance, with 30 VMs, DI with HCA is closest to 0. Hence, HCA provides a best load balance with the increasing number of heterogeneous VMs. In addition, it can provide high availability of resources.

In Fig. 6(b), HCA outperforms FSA, RB2B and RR algorithms in term of resource utilization by maximizing the utilization of resources at value closest to 100 %. This means that any resource is not under-utilized. So, this is a benefit for providers to keep resources more busy and earn maximum profit by renting limited number of resources.

Table 6 shows that makespan is decreasing when the number of VM increases due to the distribution of cloudlets among all VMs. However, we observe that the proposed HCA algorithm outperforms FSA, RB2B and RR. Comparing with RB2B and RR, HCA can even use 15 VMs to run 1,000 cloudlets in short time of 581 sec, while RB2B and RR need more than 20 VMs. Thus, HCA well minimizes makespan despite the increasing number of VMs that is an advantage of coping with energy issue in the cloud computing. However, as compared to the experiment 1 in Section IV (C), HCA produces waiting time which is little greater than FSA, as shown by Table 6, this experiment used 1,000 small cloudlets.

Through this experiment and its analysis, we concluded that the proposed HCA algorithm improves much better the utilization of resources and maximizes load balancing despite number or capacity of resources, so that cloud provider can be well satisfied.

TABLE 7. Running time of HCA under different configurations. (a) Number of VMs=20, Cloudlet length=1000-100,000 (MI). (b) Number of VMs=20, Cloudlet length =1000,000-2000,000 (MI). (c) Number of cloudlets=1000. (d) Number of VMs=2000.

(a)

Number of cloudlets	200	400	600	800	1000
RR	46	76	109	142	170
RB2B	*41*	*73*	*108*	*137*	*171*
FSA	41	71	105	138	166
HCA	**41**	**69**	**100**	**135**	**164**
Improvement	0%	2.81%	4.76%	2.17%	1.20%

(b)

Number of cloudlets	1000	2000	3000	4000	5000
RR	160	315	472	652	816
RB2B	*177*	*345*	*523*	*714*	*905*
FSA	165	325	500	691	897
HCA	**169**	**325**	**492**	**664**	**842**
Improvement	-2.42%	0%	1.60%	3.90%	6.13%

(c)

Number of VM	5	10	15	20	30
RR	158	159	161	156	160
RB2B	*159*	*163*	*161*	*161*	*164*
FSA	163	165	160	157	164
HCA	**159**	**159**	**160**	**159**	**162**
Improvement	2.45%	3.63%	0%	-1.27%	1.21%

(d)

Workload	W01	W02	W03	W04	W05
RR	332	697	951	4073	8228
RB2B	*332*	*616*	*986*	*3094*	*7483*
FSA	820	1575	2056	6055	14121
HCA	**277**	**547**	**750**	**2172**	**5413**
Improvement	16.56%	11.20%	23.93%	29.79%	27.66%

G. EFFICIENCY ANALYSIS

This section analyzes the efficiency of the proposed algorithm in term of running time with varying number of small cloudlets, large cloudlets, virtual machines and real workloads.

Concerning running time, in Table 7 (a), we observe that the result in HCA algorithm is a little less than other algorithms, RR, FSA and RB2B. This means that HCA runs faster than RR, FSA and RB2B in case of small cloudlets. However, in Table 7 (b), it shows that HCA only runs faster than FSA

and RB2B with the increasing number of large cloudlets, but its running time is greater than RR.

In Table 7 (c), we observe that the result in HCA is closest to FSA, RB2B and RR algorithms. This means that HCA runs as fast as FSA, RB2B and RR with different number of virtual machines. However, Table 7 (d) shows that HCA produces high minimum improvement than FSA, RB2B and RR with real workloads. This means that our proposed HCA algorithm runs faster than FSA, RB2B and RR.

Through this analysis of efficiency of different algorithms based on running time, we can conclude that the proposed HCA algorithm is efficient and can produce best running time to satisfy cloud providers. This is because the time complexity of HCA algorithm which has presented as equal to $O(n)$ in Section III (G), depends on the number of cloudlets and their position in the virtual machines.

V. CONCLUSION

This paper presents a heuristic cloudlet allocation approach (HCA) which improves load balancing and task scheduling in cloud resource management under the environmental changing of the number of VMs, small cloudlets (tasks), large cloudlets and real workloads. The main innovation is the combination of optimal completion time and earliest finish time which implies the combination of completion time of virtual machine and earliest finish time based on execution time of a cloudlet. In addition, optimal completion time has been introduced to ensure a best load balancing while earliest finish time was for minimizing overall completion time and maximizing resource utilization. The proposed HCA works into two phases that are as follows: allocation based on optimal completion time and allocation based on earliest finish time. The detailed experimental results demonstrate that the proposed HCA provides a best load balancing, highest resource utilization and lowest makespan in comparison with FSA, RB2B and RR algorithms. Further, the results of running time validate the efficiency of HCA. In the near future, we intend to extend our approach for live migration and security problem and on real cloud infrastructures.

REFERENCES

[1] A. A. Buhussain, R. E. De Grande, and A. Boukerche, "Elasticity based scheduling heuristic algorithm for cloud environments," in *Proc. IEEE/ACM 20th Int. Symp. Distrib. Simulation Real Time Appl.*, London, U.K., Sep. 2016, pp. 1–8.

[2] S. R. Shishira, A. Kandasamy, and K. Chandrasekaran, "Survey on meta heuristic optimization techniques in cloud computing," in *Proc. IEEE Int. Conf. Adv. Comput., Commun. Informat.*, Jaipur, India, Sep. 2016, pp. 1434–1440.

[3] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, Melbourne, VIC, Australia, May 2010, pp. 826–831.

[4] Y. Chen, G. Xie, and R. Li, "Reducing energy consumption with cost budget using available budget preassignment in heterogeneous cloud computing systems," *IEEE Access*, vol. 6, pp. 20572–20583, 2018.

[5] F. Juarez, J. Ejarque, and R. M. Badia, "Dynamic energy-aware scheduling for parallel task-based application in cloud computing," *Future Gener. Comput. Syst.*, vol. 78, pp. 257–271, Jan. 2018.

[6] H. Liu, F. Yan, S. Zhang, T. Xiao, and J. Song, "Source-level energy consumption estimation for cloud computing tasks," *IEEE Access*, vol. 6, pp. 1321–1330, 2017.

[7] R. Jithin and P. Chandran, "Virtual machine isolation: A survey on the security of virtual machines," in *Proc. Int. Conf. Secur. Comput. Netw. Distrib. Syst.*, 2014, pp. 91–102.

[8] D. Chaudhary and B. Kumar, "An analysis of the load scheduling algorithms in the cloud computing environment: A survey," in *Proc. 9th IEEE Int. Conf. Ind. Inf. Syst.*, Gwalior, India, Dec. 2014, pp. 1–6.

[9] S. Banerjee, M. Adhikari, S. Kar, and U. Biswas, "Development and analysis of a new cloudlet allocation strategy for QoS improvement in cloud," *Arabian J. Sci. Eng.*, vol. 40, no. 5, pp. 1409–1425, 2015.

[10] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[11] S. H. H. Madni, M. S. A. Latiff, M. Abdullahi, S. M. Abdulhamid, and M. J. Usman, "Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment," *PLoS One*, vol. 12, no. 5, p. e0176321, 2017.

[12] S. Roy, S. Banerjee, K. R. Chowdhury, and U. Biswas, "Development and analysis of a three phase cloudlet allocation algorithm," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 29, no. 4, pp. 473–483, 2017.

[13] E. I. Djebbar and G. Belalem, "Tasks scheduling and resource allocation for high data management in scientific cloud computing environment," in *Mobile, Secure, and Programmable Networking* (Lecture Notes in Computer Science), vol. 10026. New York, NY, USA: Springer-Verlag, 2016, pp. 16–27.

[14] H. B. Alla, S. B. Alla, and A. Ezzati, "A novel architecture for task scheduling based on dynamic queues and particle swarm optimization in cloud computing," in *Proc. 2nd IEEE Int. Conf. Cloud Comput. Technol. Appl.*, Marrakech, Morocco, May 2016, pp. 108–144.

[15] R. K. Jena, "Multi objective task scheduling in cloud environment using nested PSO framework," in *Proc. 3rd Int. Conf. Recent Trends Comput.*, Delhi, India, 2015, pp. 1219–1227.

[16] A. Khalili and S. M. Babamir, "Makespan improvement of PSO-based dynamic scheduling in cloud environment," in *Proc. 23rd IEEE Iranian Conf. Elect. Eng.*, Tehran, Iran, May 2015, pp. 613–618.

[17] M. Masdari, F. Salehi, M. Jalali, and M. Bidaki, "A survey of PSO-based scheduling algorithms in cloud computing," *J. Netw. Syst. Manage.*, vol. 25, no. 1, pp. 122–158, 2017.

[18] P. Singh, M. Dutta, and N. Aggarwal, "A review of task scheduling based on meta-heuristics approach in cloud computing," *Knowl. Inf. Syst.*, vol. 52, no. 1, pp. 1–51, 2017.

[19] S. Singh and A. Ranjan, "An improved task scheduling algorithm based on PSO for cloud computing," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 5, no. 11, pp. 16773–16777, 2017.

[20] M. B. Gawali and S. K. Shinde, "Task scheduling and resource allocation in cloud computing using a heuristic approach," *J. Cloud Comput.*, vol. 7, no. 4, pp. 1–16, 2018.

[21] W. Hashem, H. Nashaat, and R. Rizk, "Honey bee based load balancing in cloud computing," *Trans. Internet Inf. Syst.*, vol. 11, no. 12, pp. 5694–5711, 2017.

[22] M. B. Gawali and S. K. Shinde, "Standard deviation based modified cuckoo optimization algorithm for task scheduling to efficient resource allocation in cloud computing," *J. Adv. Inf. Technol.*, vol. 8, no. 40, pp. 210–218, 2017.

[23] E. J. Ghomia, A. M. Rahmania, and N. N. Qader, "Load-balancing algorithms in cloud computing: A survey," *J. Netw. Comput. Appl.*, vol. 88, pp. 50–71, Jun. 2017.

[24] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, "Cost-efficient task scheduling for executing large programs in the cloud," *Parallel Comput.*, vol. 39, pp. 177–188, Apr./May 2013.

[25] R. J. Priyadarsini and L. Arockiam, "Performance evaluation of min-min and max-min algorithms for job scheduling in federated cloud," *Int. J. Comput. Appl.*, vol. 99, no. 18, pp. 47–54, 2014.

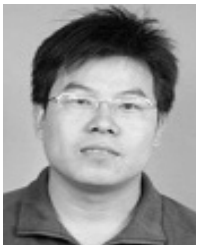
[26] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.

[27] P. Humane and J. N. Varshapriya, "Simulation of cloud infrastructure using cloudsim simulator: A practical approach for researchers," in *Proc. Int. Conf. Smart Technol. Manage. Comput., Commun., Controls, Energy Mater.*, May 2015, pp. 207–211.

[28] Q.-Y. Chen, Z.-H. Liang, H.-W. Kang, Y.-M. Ma, and D. Wang, "Research of dependent tasks scheduling algorithm in cloud computing environments," in *Proc. 3rd Annu. Int. Conf. Inf. Technol. Appl.*, vol. 7, pp. 1–6, Nov. 2016.

[29] S. J. Chapin *et al.*, "Benchmarks and standards for the evaluation of parallel job schedulers," in *Job Scheduling Strategies for Parallel Processing* (Lecture Notes in Computer Science), vol. 1659, D. G. Feitelson and L. Rudolph, Eds. New York, NY, USA: Springer-Verlag, 1999, pp. 66–89. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html

JEAN PEPE BUANGA MAPETU received B.S. degree in computer science from Kinshasa University, Democratic Republic of Congo, in 2006, and the M.S. degree in computer science and engineering from Yanshan University, China, in 2016, where he is currently pursuing the Ph.D. degree with the College of Computer Science and Engineering. He is currently working on cloud computing and parallel and distributed computing.



ZHEN CHEN received the B.S. and Ph.D. degrees in computer application technology from Yanshan University, China, in 2010 and 2017, respectively. He is currently a Post-Doctoral Researcher with the College of Computer Science and Engineering, Yanshan University. He is currently working on service computing, cloud computing, and collaborative computing.

LINGFU KONG received the Ph.D. degree in computer science and technology from the Harbin Institute of Technology University, China, in 1995. He is currently a Professor and a Ph.D. Supervisor with the College of Computer Science and Engineering. His main research interests include robot, computer vision, intelligent information processing, and cloud computing.