The Journal of Supercomputing
https://doi.org/10.1007/s11227-019-02916-4

SLA-RALBA: cost-efficient and resource-aware load balancing algorithm for cloud computing

Altaf Hussain[1] · **Muhammad Aleem**[1] · **Muhammad Azhar Iqbal**[1] · **Muhammad Arshad Islam**[1]

© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract

Recently, *Service-level agreement* (SLA) is deemed to be an integral aspect for on-demand provisioning of scalable resources on Cloud. SLA defines important constraints for instance guaranteed *quality of service* (QoS), pricing, fault-tolerant availability, security, and period of service. Currently, there is a dire need of SLA-based scheduling that improves resources utilization on Cloud. Scheduling with reduced execution time and cost may adversely affect resource utilization. To overcome this issue, we present a cost-efficient SLA-based load balancing scheduler, namely SLA-RALBA, for heterogeneous Cloud infrastructures. The proposed technique supports three levels of SLA opted by the Cloud users. The proposed novel technique incorporates the execution cost for the successful execution of users' services to elevate the resource utilization on Cloud. The SLA-RALBA is simulated for performance analysis using the benchmark GoCJ and HCSP datasets. The performance results of the SLA-RALBA are compared with the existing schedulers, namely Execution-MCT, Profit-MCT, SLA-MCT, Execution-Min-Min, Profit-Min-Min, and SLA-Min-Min in terms of average resource utilization, execution time, and cost of the Cloud services. The obtained results reveal that SLA-RALBA provides an even trade-off between execution time and cost of the services by guaranteeing a drastic improvement in resource utilization on Cloud than existing algorithms.

Keywords RALBA · Service-level agreement · Cost-efficient scheduling · Resource-aware scheduling

✉ Altaf Hussain
saddamaltaf@gmail.com

[1] Department of Computer Science, Faculty of Computing, Capital University of Science and Technology, Islamabad, Pakistan

Published online: 05 June 2019

1 Introduction

Cloud datacenters are generally over-provisioned to guarantee high service availability and QoS computing [1]. The QoS contracts are formally negotiated and self-proclaimed in SLA to ensure the execution and cost constraints committed to the Cloud users [2, 3]. Over-provision of resources by Cloud datacenters adversely affects the resource utilization and leads to imbalance distribution of workload by generating stragglers [4]. The straggling or slowly running jobs can run up to eight times slower than the median task on a production cluster that increases the job execution time up to 47% [5, 6]. If the adopted scheduling strategies result in a workload imbalance with stragglers (i.e., slowly executing jobs), then various machines remain idle or underutilized in a heterogeneous computing environment. Therefore, an increasing amount of straggling jobs in workload distribution reduces the revenue generation of *Cloud Service Provider* (CSP) in terms of workload execution gain [7]. The fundamental challenge associated with straggling jobs is faced by all Cloud frameworks [6] (e.g., MapReduce [8], Dryad [9], and Spark [10]). Therefore, each CSP is elicited to be equipped with a growing resource capacity to fulfill the dynamic and drastically increasing demands of Cloud users [3, 11]. However, the straggling, as well as the improper distribution of workload especially in case of SLA-based scheduling, often results in lower resource utilization in Cloud datacenters [7, 11, 12].

This study focuses on the problem of low-resource utilization in SLA-aware Cloud scheduling and presents load balancing Cloud scheduler named *service-level agreement-based resource-aware load balancing algorithm* (SLA-RALBA) to mitigate the load imbalance issue. SLA-RALBA is a novel scheduling heuristic that schedules the SLA-based jobs in a load-balanced manner to improve resource utilization. SLA-RALBA is an extension of existing load balancing scheduler RALBA [13]. Unlike RALBA, the SLA-RALBA considers execution cost as an additional parameter. SLA-RALBA is a load balancing technique proposed for SLA-based compute-intensive, non-preemptive, independent, execution time, and cost-cognizant Cloud jobs. The primary objective of SLA-RALBA is to maximize the resource utilization and minimize the execution time and cost. The SLA-RALBA comprises of two sub-schedulers; *Fill* and *Spill* Schedulers. The workload is scheduled on *Virtual Machines* (VMs) according to the SLA type and computing requirements of Cloud jobs in the workload by the *Fill* sub-scheduler. The *Spill* sub-scheduler schedules the remaining jobs on VMs by producing the *Earliest Finish Time* (EFT) in addition to considering the execution cost. Performance evaluation of SLA-RALBA reveals that the proposed load balancing scheduler attains up to 12.09 times increased resource utilization as compared to existing state-of-the-art schedulers, such as Profit-MCT, Profit-Min-Min, and SLA-Min-Min heuristics (using the instances of benchmark *Google Cloud Jobs* (GoCJ) dataset [14]). However, the SLA-RALBA attains up to 14.51% improved resource utilization as compared to SLA-Min-Min and up to 154.55% as compared to Profit-Min-Min and Profit-MCT (using instances of benchmark *Heterogeneous Computing Scheduling Problems* (HCSP) dataset [15, 16]). The major contributions of proposed study are as follows:

- In-depth scrutiny of the literature to analyze strengths and weaknesses of the existing scheduling heuristics;
- Cost-efficient SLA-RALBA scheduling mechanism for SLA-based compute-intensive, non-preemptive, and independent jobs that produces an improved balance among execution time, cost, and resource utilization, and
- Performance comparison of the proposed SLA-RALBA scheduler with existing scheduling heuristics using 15 instances of two benchmark datasets, i.e., GoCJ [14] and HCSP [15, 16], for seven different computing architecture on CloudSim [17].

The rest of paper is organized as follows. Section 2 presents related work. Section 3 presents the system architecture, system and performance model, and the algorithm of the proposed scheduling technique. Section 4 describes the experimental environment, benchmark workloads, and discussion on the attained performance results. Section 5 presents the concluding discussion with future directions.

2 Related work

To date, various studies have been conducted that focus on budget, user priorities, execution time, response time, deadline, and SLA violations as QoS requirements for Cloud resource provisioning and scheduling.

Garg et al. proposed *Mixed Workload-Aware Policy* (MWAP) [12] that is an admission control and scheduling technique to enhance the resource utilization and profit of Cloud datacenters while guaranteeing the QoS needs of Cloud users as per SLA. MWAP employed *Artificial Neural Network* (ANN) to forecast the future computing utilization of the datacenters.

Philipp Leitner et al. proposed SLA-aware scheduling technique to reduce the expenses on Cloud resources and loss incurred as SLA violations in an *Infrastructure-as-a-Service* (IaaS) Cloud [18]. This cost-efficient technique ensures decreased cost consumption of CSP.

Alrokayan et al. proposed a cost and SLA-aware Branch and *Bound algorithm for a Pruned tree* (BBPruned) [19] technique for big data analytics. The BBPruned allocates Cloud resources for scheduling of MapReduce tasks with a budget and deadline constraints without SLA violation. Sharma et al. presented a technique named as Kingfisher [20] that is a cost-efficient framework offering effectual provision of elasticity on Cloud to minimize the incurred cost. Kingfisher supports single application provisioning while ignoring the SLA and mixed workload. Lenzini proposed *Active List Queue Method* (ALiQueM), which is an implementation of *Deficit Round-Robin* (DRR) scheduling to guarantee the significant trade-off between latency and fairness with low complexity of DRR [21]. ALiQueM proves its efficacy against its counterparts such as Pre-Order DRR and Smoothed DRR scheduling techniques.

Execution Minimum Completion Time (Execution-MCT) allocates a job to the VM producing minimum completion time for it [16, 22–24]. On each scheduling decision, the current load of a VM is considered to identify an appropriate VM for the allocation of a candidate job [16, 25]. The VM producing minimum

completion time for a job is selected and assigned to it. The execution-MCT overloads faster as compared to the slower VMs that result in load imbalance [26, 27]. Execution-Min-Min heuristic operates on the scheduling mechanism of execution-MCT [24, 28, 29] that works in two scheduling steps: (1) The earliest completion time of every job is calculated considering all the VMs and (2) the job with minimum of these completion times is identified and allocated to a concerned VM. Execution-Min-Min favors smaller jobs and penalizes larger jobs [25, 30]. The execution-Min-Min introduces a load imbalance by overloading faster VMs with smaller jobs [13, 16]. *Priority-Aware Load-Balanced Improved Min–Min* (PA-LBIMM) categorized the jobs and computing machines into VIP and normal classes [29]. The jobs of the former class are allocated to VIP machines using Execution-Min-Min. Afterward, jobs of the later class are scheduled on all computing machines using the Execution-Min-Min.

Profit-MCT and Profit-Min-Min [31] heuristics are based on execution cost despite execution time. The scheduling mechanisms of Profit-MCT and Profit-Min-Min are identical to the Execution-MCT and Execution-Min-Min, respectively, with a difference in the objective functions (i.e., minimization of execution time in Execution-MCT and the minimization of execution cost in Execution-Min-Min heuristic). On each scheduling step, the jobs are assigned to the VM that executes it with the minimum execution cost. Both the Profit-MCT and Profit-Min-Min overload the VMs with the least execution cost and result in a load imbalance due to the unutilized VMs [31]. SLA-Min-Min and SLA-MCT [31] are based on common SLA to form a balance between the overall execution time and cost. SLA-Min-Min and SLA-MCT use three types of SLA levels: minimization of execution time, cost, and both (i.e., *time* and *cost*), for the submitted workload. The SLA-1 jobs prefer to be executed within minimal execution time, while the SLA-3-based jobs prefer to be executed with minimal execution cost, and the SLA-2-based jobs tend to execute with a balance between both execution time and cost. Let us consider that there is SLA-2 job that has execution time 22, 8, and 15 s on VM_1, VM_2, and VM_3, respectively. Similarly, job_1 has execution cost 2, 8, and 6 unit costs on VM_1, VM_2, and VM_3, respectively. For SLA-02 jobs, the weight values for both the execution time and execution cost will be 0.5 (i.e., $w_1 = w_2$). However, different weights may also be given by the user for certain SLA jobs. For SLA-based jobs with weight values for execution time and execution cost, SLA-MCT calculates the sum of execution time and cost on all the VMs. The execution time and cost are normalized by dividing with corresponding sum values. Thereafter, this heuristic finds the weighted sum by multiplying the normalized execution time and execution cost value with the corresponding weighted values given by the user for a job. Table 1 presents an example to illustrate the mapping of SLA-2 job to VM. The VM-1 has smaller weighted sum 0.23 for SLA-2 job so that the job is scheduled on VM-1. However, the SLA-1 jobs are scheduled on VM producing minimum completion time and SLA-3 jobs are scheduled on VM producing minimum cost value. Table 2 recapitulates existing scheduling heuristics.

Table 1 Example of SLA-2 job with w1 = 0.3 and w2 = 0.7 values for job—VM mapping

	VM-1	VM-2	VM-3
	Execution time	**Execution time**	**Execution time**
Job →	22	8	15
Weighted value	22/45 = **0.49**	8/45 = **0.18**	15/45 = **0.34**
	Execution cost	**Execution cost**	**Execution cost**
Job →	2	8	6
Weighted value	2/16 = **0.13**	8/16 = **0.5**	6/16 = **0.38**
	Weighted sum	**Weighted sum**	**Weighted sum**
Weighted sum	**0.49***0.3 + **0.13***0.7 = 0.23	**0.18***0.3 + **0.5***0.7 = 0.40	**0.34***0.3 + **0.38***0.7 = 0.36

Bold indicate that the weighted values of execution time and cost that will affect the VM selection for mapping for SLA-2 jobs

3 Cost-efficient resource-aware load balancing algorithm

This section delineates the proposed cost-efficient and resource-aware scheduling technique named SLA-RALBA. The system overview, performance model, and the algorithm of SLA-RALBA are explained in this section.

3.1 SLA-RALBA overview

SLA-RALBA is a cost-efficient load balancing algorithm consisting of two sub-schedulers similar to RALBA [13], which are *Fill* and *Spill* schedulers. Based on Cloud computing architecture, an abstract level of SLA-RALBA is shown in Fig. 1. In this study, we employed CloudSim simulator. The user job is known as a cloudlet [17]. The virtual instance layers are hosted by physical instance layer that provides the foundation for the delivery of SLA-aware computing services to the Cloud users [31]. The Cloud datacenters contain an assortment of physical host machines and storage media at the physical instance layer. The computing, communication, and storage resources are provisioned on-demand in a scalable and dynamic manner to the Cloud users in the form of virtual machines formulating the visual instance layer. The resource manager, on the top of the virtualization layer, is responsible for creation, termination, and migration of VMs when required. In addition, the resource manager and cloudlet scheduler allocate VMs on requirement and record the cost estimation of the provisioned services. Therefore, the resource manager provides the information pertaining to the available VMs that are the computing capabilities and the incurred gain cost of each VM. To ensure the utilization of VMs with its full computing capacity and minimal execution cost, a cost-efficient and resource-aware load balancer is required on top of the Cloud virtualization for balanced workload distribution.

The system accessibility layer of SLA-RALBA provides a user-friendly interface to submit SLA-aware cloudlets on Cloud. The SLA-RALBA scheduler is

Table 2 Summary of the related scheduling heuristics

Heuristics	Strengths	Weaknesses
SLA-MCT [31]	Cost-efficient scheduling, single-phase scheduling to balance between execution time and cost [31]	Poor resource utilization, load imbalance [31]
SLA-Min-Min [31]	Cost-efficient scheduling, two-phase scheduling to balance between execution time and cost [31]	Poor resource utilization, load imbalance [31]
Profit-MCT [31]	Cost-efficient scheduling [31]	Overloads machines with least execution cost [31], poor resource utilization,
Profit-Min-Min [31]	Cost-efficient scheduling [31]	Overloads machines with least execution cost [31], poor resource utilization
Execution-MCT [7, 13]	Improved execution time for jobs, improved makespan than traditional round-robin technique [13]	Overloads faster machines do not consider execution cost [31]
Execution-Min-Min [7, 13]	Favors small-sized jobs, reduced turnaround time for small-sized jobs [13]	Penalizes large-sized jobs, overloads faster machines with small-sized jobs do not consider execution cost [31]

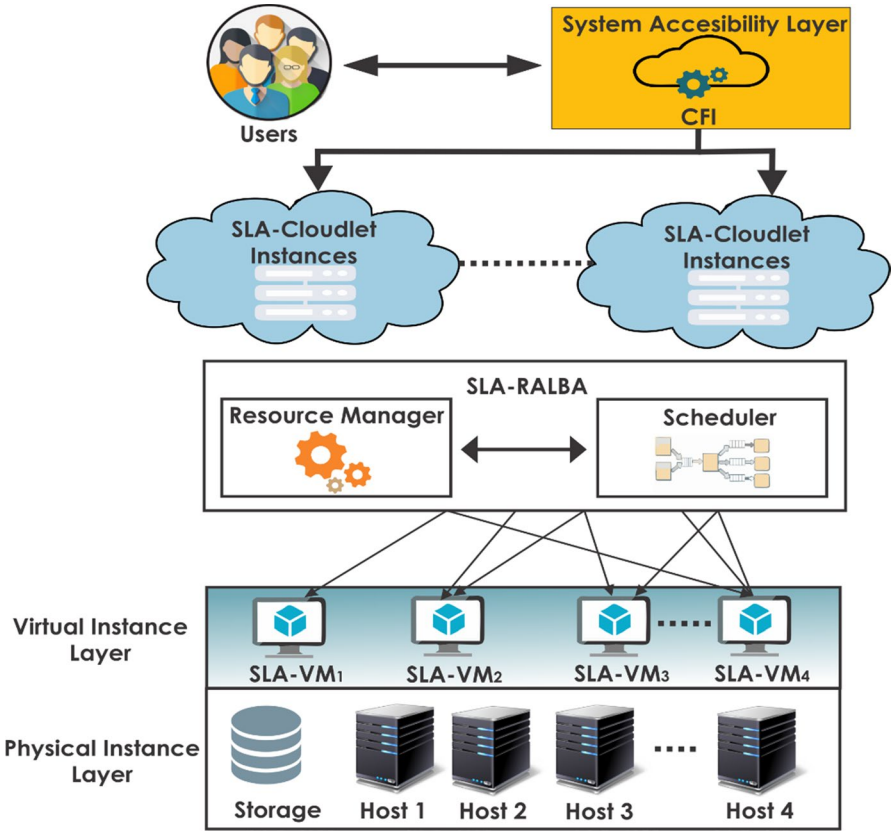


Fig. 1 Abstraction layer of SLA-RALBA

proposed on top of the SLA-aware virtualization to improve the resource utilization with a cost-efficient and load-balanced schedule ensuring the minimal execution time and cost (as shown in Fig. 1).

3.2 SLA-RALBA system architecture

The system and performance model of the proposed technique are presented using mathematical expressions. For better comprehension, majority of the basic definitions and terminologies used in RALBA are repeated for the representation of SLA-RALBA. The basic definitions and notations used in the mathematical expressions are illustrated in Table 3.

The system architecture of SLA-RALBA is shown in Fig. 2. SLA-RALBA comprises two sub-schedulers, i.e., *Fill* and *Spill* schedulers. *Fill* scheduler performs cloudlet to VM allocation by considering the computing share of VMs. *Fill* scheduler selects VM_j with the *Largest_VMShare* and determines the *maxPCloudlet* for this VM_j. The *maxPCloudlet* is checked for its SLA type (i.e., SLA-1, SLA-2, or

Table 3 Preliminary notations used in SLA-RALBA system and performance models

Notations	Descriptions
vmCrMap	Set of sorted VMs with its computing ratio
$vmCrMap_j$	Computation ratio of VM *j* in *vmCrMap*
$RPCloudlet_j$	Set of remaining possible cloudlets that can be assigned to VM_j
$maxPCloudletVM_j$	Largest cloudlet in $RPCloudlet_j$ that is assigned to VM_j
$Cloudlet.\text{EGain}_{ij}$	Execution cost paid to CSP for executing $Cloudlet_i$ on VM_j
$VM_j.ECost$	Execution cost of VM_j per unit execution time
VMShare	Set of sorted VMs with its computing share
$VMShare_j$	Computing share of VM *j* (in MI) in *VMShare*
$minEGain_i$	VM_j with $VMShare_j \geq Cloudlet_i.MI$ and least EGain for $Cloudlet_i$
VM_EGain_j	Total EGain of VM_j by executing its all allocated cloudlets
Largest_VMShare	Current largest computing share for any VM in *VMS*

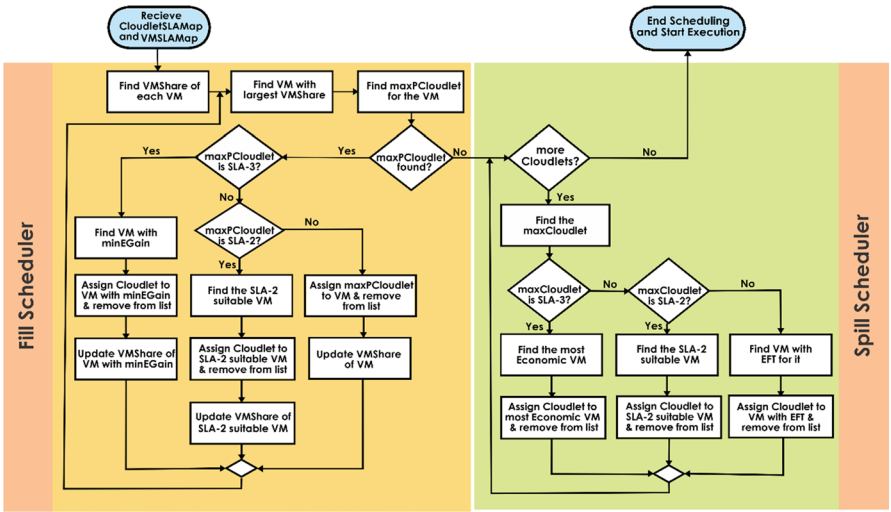


Fig. 2 SLA-RALBA system architecture

SLA-3). If the maxPCloudlet is SLA-3, then the VM with *minEGain* is determined for it. The maxPCloudlet is assigned to the VM with minEGain and removed from the list of cloudlets. The VMShare of the VM with minEGain is also updated. If the maxPCloudlet is SLA-2 based, then the suitable VM for SLA-2 is determined for it. The maxPCloudlet is assigned to this SLA-2 suitable VM and removed from the list of cloudlets. The VMShare of the SLA-2 VM is also updated. If the maxPCloudlet belongs to SLA-1, then the maxPCloudlet is allocated to the VM_j with Largest_VMShare and removed from the list of cloudlets and VMShare of VM_j is updated accordingly. *Fill* scheduler repeats the process of cloudlets mapping to VMs until there does not exist maxPCloudlet for any of the VMs in Cloud datacenter.

Spill scheduler performs cloudlets to VMs allocation based on descending order of cloudlet sizes. The largest sized cloudlet (as maxCloudlet) is selected and is checked

for SLA type. If the maxCloudlet is SLA-3 based, then the maxCloudlet is allocated to the economical VM (mapping similar to *Fill* scheduler) and removed from the list of cloudlets. If the maxCloudlet is SLA-2, then the maxCloudlet is assigned to the SLA-2 suitable VM and removed from the list of cloudlets. If the maxCloudlet belongs to SLA-3, then the maxCloudlet is assigned to the VM producing EFT and removed from the cloudlets list. *Spill* scheduler repeats the Cloudlets to VMs mapping until the list of cloudlets becomes empty.

3.3 System model

The Cloud has a service manager responsible for providing a unified service to Cloud users over the Internet. The service manager has knowledge of the Cloud computation resources, and the services are delivered to the Cloud user by deploying VMs in the datacenters. The Cloud system comprises a set of VMs that is represented as $VMs = \{VM_1, VM_2, \ldots, VM_m\}$, where m is the total number of VMs and $VM_j (1 \leq j \leq m)$ stands for VM number j. The VM is measured in terms of *Million Instructions Per Second* (MIPS). VM_j. MIPS represents the MIPS of VM_j. However, the *Expected Gain* (EG_j) per unit time is the *Gain Cost* (GC) of executing user request on a specific VM_j in Cloud. The user request on the Cloud is known as a cloudlet. The set of cloudlets submitted by users for execution with some SLA levels is $CLS = \{Cloudlet_1, Cloudlet_2, \ldots, Cloudlet_n\}$, where n is the number of cloudlets, and $Cloudlet_i (1 \leq i \leq n)$ stands for cloudlet number i. The cloudlet is measured in terms of *Million Instructions* (MI) and $Cloudlet_i$. MI represents MIs of $Cloudlet_i$. Each cloudlet in the CLS belongs to one of the three SLA levels: SLA-1, SLA-2, and SLA-3 as used in [31]. The SLA-1 only focuses on the execution time; however, SLA-3 only focuses on the execution cost of the cloudlets. The proposed SLA-2 is a combination of SLA-1 and SLA3, i.e., considering both the execution time and cost of the cloudlet scheduling.

The computation share of any VM_j is represented as $VMShare_j$ and mathematically expressed as:

$$VMShare_j = \left(\sum_{i=1}^{n} Cloudlet_i.MI \right) \times \left(\frac{VM_j.MIPS}{\sum_{k=1}^{m} VM_k.MIPS} \right) \quad \forall j = \{1, 2, \ldots, m\} \tag{1}$$

The list of remaining possible cloudlets could be allocated to a particular VM_j, which is represented as $RPCloudlet_j$ and mathematically expressed in Eq. 2. The largest possible cloudlet for VM_j in $RPCloudlet_j$ is represented as $maxPCloudlet4VM_j$ and expressed in Eq. 2:

$$maxPCloudlet4VM_j = \max_{\forall p \in RPCloudletj} f(p.MI) \tag{2}$$

where

$$RPCloudlet_j = \{Cloudlet | \forall Cloudlet \in CLS, Cloudlet.MI \leq VMShare_j\} \tag{3}$$

The VM_j with *minEGain* for $Cloudlet_i$ is determined with the following relation.

$$minEGain_i = \min_{\forall v \in VMS \wedge Cloudlet_i.MI \leq VMShare_v} f\left(Cloudlet.EGain_{iv}\right) \tag{4}$$

The EGain of $Cloudlet_i$ on a particular VM_j is determined with the following relation.

$$Cloudlet.EGain_{ij} = VM_j.ECost \times \frac{Cloudlet_i.MI}{VM_j.MIPS} \tag{5}$$

The most economic VM_j for $Cloudlet_i$ is the virtual machine that produces minimum cost to execute $Cloudlet_i$.

$$mostEconomicVM_i = \min_{\forall v \in VMS} f\left(Cloudlet_i.EGain_{iv}\right) \tag{6}$$

The set of cloudlets to be scheduled is presented as *RCLS* that is updated after each scheduling step considering the following relation:

$$RCLS = \begin{cases} (CLS) - (maxPCloudlet4VMj), & First\ Cloudlet\ assignemt \\ (RCLS) - (maxCloudlet4VMj), & Using\ Fill - Scheduler \\ \\ (RCLS) - (maxCloudlet), & Using\ Spill - Scheduler \end{cases} \tag{7}$$

The size of cloudlet is measured in MIs; therefore, the minCloudlet and maxCloudlet is determined in terms of its MIs (i.e., If $Cloudlet_1.MI = 250$, $Cloudet_2.MI = 300$, and $Cloudlet_3.MI = 500$, then Cloudlet_1 and Clouldlet_3 are *minCloudlet* and *maxCloudlet*, respectively). The smallest and largest cloudlets from the remaining list of cloudlets are selected and mathematically expressed as:

$$minCloudlet = \begin{cases} \min\limits_{\forall j \in CLS} f(j.MI), & First - time \\ \min\limits_{\forall j \in RCLS} f(j.MI), & Otherwise \end{cases} \tag{8}$$

$$maxCloudlet = \begin{cases} \max\limits_{\forall j \in CLS} f(j.MI), & First - time \\ \max\limits_{\forall j \in RCLS} f(j.MI), & Otherwise \end{cases} \tag{9}$$

$VMShare_j$ is modified in each step of *Fill scheduler* using Eq. (10) and *Largest_VMShare* is determined using Eq. (11).

$$VMShare_j = \begin{cases} Switch\ to\ SpillScheduler, VMShare_j < minCloudlet \\ VMShare_j - maxPCloudlet4VM_j.MI, VMShare_j \geq minCloudlet \end{cases} \tag{10}$$

$$Largest\_VMShare = \max_{\forall j \in \{1,2,3,\ldots,m\}} \left(VMShare_j\right) \tag{11}$$

Fill scheduler selects the VM with *Largest_VMShare* in each step of cloudlet to VM scheduling.

3.4 Performance model

The completion time of $Cloudlet_i$ on VM_j is mathematically defined as [13]:

$$Cloudlet\_CT_{ij} = \left(\frac{Cloudlet_i.MI}{VM_j.MIPS}\right) + VM\_CT_j \tag{12}$$

The *Earliest Finish Time* (EFT) for $Cloudlet_i$ on any available VM is defined as [13]:

$$Cloudlet\_EFT_i = \min_{\forall j \in \{1,2,3,\ldots,m\}} \left(Cloudlet\_CT_{ij}\right) \tag{13}$$

Let *nj* be the number of cloudlets assigned to a VM_j and F[i,j] be a Boolean variable, that is,

$$F[i,j] = \begin{cases} 1, & cloudlet\ i\ is\ assigned\ to\ VMj \\ 0, & Otherwise \end{cases} \tag{14}$$

The completion time of VM_j is expressed in Eq. 15 [13].

$$VM\_CT_j = \sum_{i=1}^{nj} (Cloudlet\_CT_{ij} \times F[i,j]) \tag{15}$$

Makespan [29, 31] and the average completion time of Cloud are represented and expressed in Eqs. 16 and 17, respectively.

$$Makespan = \max_{\forall j \in \{1,2,3,\ldots,m\}} \left(VM\_CT_j\right) \tag{16}$$

$$Avg\_VM\_CT = \frac{\sum_{j=1}^{m} VM\_CT_j}{m} \tag{17}$$

The average resource utilization ratio is mathematically presented in Eq. 18 [13, 29, 31]. The ARUR value varies between 0 and 1; a higher value of ARUR depicts high resource utilization.

$$ARUR = \frac{Avg\_VM\_CT}{Makespan} \tag{18}$$

A total Gain of particular VM_j is represented as VM_EGain_j and mathematically expressed in Eq. 19, followed by the gain of the computing system presented in Eq. 20. The gain cost is desired to be reduced for Cloud user jobs.

$$VM\_EGain_j = \sum_{i=1}^{nj} (Cloudlet.EGain_{ij} \times F[i,j]) \tag{19}$$

$$EGain = \sum_{j=1}^{m} VM\_EGain_j \tag{20}$$

3.5 SLA-RALBA algorithm

In this section, the proposed SLA-RALBA load balancing technique is explained with its two primary schedulers, i.e., *Fill* and *Spill*.

A list of Cloudlets with its SLA types is submitted by Cloud users as *cloudSLAList*. The Cloud resource manager provides the list of all VMs computation ratio as an input parameter to SLA-RALBA scheduler. Scheduling SLA-based cloudlets on VMs, SLA-RALBA invokes *Fill* scheduler (line 2 of Algorithm 1) and then invokes the *Spill* scheduler (line 5 of Algorithm 1).

The necessary initialization is accomplished by the *Fill* scheduler which computes the *totalLength* of all the SLA-based cloudlets in the submitted workload (lines 5–6 of Algorithm 2). Afterward, the computing share (vShareMapv) of each VM is determined (lines 7–8 of Algorithm 2) using Eq. 2. A while loop (lines 9–22 of Algorithm 2) is used to schedule the SLA cloudlets on VMs based on computation share and the execution cost of the VMs. On each scheduling decision, at first the VM with the largest computing share is determined and the maxPCloudletVm is selected (lines 10–11 of Algorithm 2). If the maxPCloudletVm is of SLA-3 type, then the VM selection is revised and the *VmWithMinEGain* is determined for maxPCloudlet to be mapped on it (lines 13–14 and line 17 of Algorithm 2). If the maxPCloudletVm is of SLA-2 type, then the VM selection is revised and the *SLA2SuitableVM* is determined for maxPCloudlet to be mapped on it (lines 15–16 and line 17 of Algorithm 2). If the maxPCloudletVm is of SLA-1 type, the cloudlet is mapped to the VM with the largest computation share (line 17 of Algorithm 2). On each scheduling step, when the cloudlet is assigned to a specific VM, its computing share is modified (lines 18–19 of Algorithm 2) and the cloudlet is detached from the list of cloudlets to be scheduled (line 20 of Algorithm 2). This scheduling process is repeated until no VM with a remaining computation share is found to accept any cloudlet for scheduling (in line 11 of Algorithm 2, when the maxPCloudletVm is not found and cloudlet is initialized with null).

Algorithm 1: SLA-RALBA

Input: $vmCrMap$ - set of VMs with computation ratio,
$cloudletSLAList$ - list of SLA-cloudlets $c_1, c_2, c_3, \ldots, c_n$
Output: $cloudletVmMap$ - set of cloudlets to VMs mapping

1 $cloudletVmMap = Null$
2 $cloudletVmMap =$ FillScheduler($vmCrMap$,$cloudletSLAList$)
3 $vmList =$ getVmList($vmCrMap$)
4 **if** $cloudletSLAList.size() \geq 1$ **then**
5 $\quad cloudletVmMap =$SpillScheduler($vmList$,$cloudletSLAList$,$cloudletVmMap$)
6 **return** $cloudletVmMap$

The *Spill* scheduler is engaged in the allocation of remaining cloudlets to VMs, immediately after the *Fill* scheduler completes its working. The while loop (lines 3–12 of Algorithm 3) assigns the remaining cloudlets to VMs in descending order of cloudlets sizes. On each scheduling decision, the maxCloudlet is determined (line 4 of Algorithm 3) and scheduled on VMs. If the maxCloudlet is of SLA-3 type, then the maxCloudlet is scheduled on most economic VM (lines 5–6 and lines 11–12 of Algorithm 3). If the maxCloudlet is of SLA-2 type, then the maxCloudlet is scheduled on the suitable VM (a VM that provides the trade-off between execution time and execution cost (lines 7–8 of Algorithm 3). If the maxCloudlet is of SLA-1 type, then it is allocated to the VM that produces the earliest finish time for it (lines 9–12 of Algorithm 3). On each scheduling step, the maxCloudlet is scheduled on a candidate VM and detached from the cloudlets to be scheduled. This scheduling process is repeated until cloudletSLAList is empty.

Algorithm 2: FillScheduler

Input: $vmCrMap$ - set of VMs with computation ratio ,
$cloudletSLAList$ - list of cloudlets $c_1, c_2, c_3, \ldots, c_n$
Output: $cloudletVmMap$ - set of cloudlets to VMs mapping

```
1  totalLength = 0
2  newVShare = 0
3  cloudletVmMap = Null
4  vShareMap < v, share >= Null
5  forall cloudlet in cloudletSLAList do
6      totalLenght = totalLenght + cloudlet.getCloudletLength()
7  forall v in vmList do
8      vShareMap_v = totalLenght * vmCrMap_v
9  while cloudletSLAList.size() ≥ 1 do
10     v = getLargeShareVm(vShareMap)
11     cloudlet = getMaxPCloudletVm(v,cloudletList)
12     if cloudlet != Null then
13         if cloudlet is SLA-3 then
14             v = getVMWithMinEGain(cloudlet,vmList)
15         else if cloudlet is SLA-2 then
16             v = getSLA2SuitableVM(cloudlet,vmList)
17         cloudletVmMap.add(cloudlet,v)
18         newVShare = vShareMap.get(v)− cloudlet.getCloudletLength
19         vShareMap.modify(v,newVShare)
20         cloudletList.remove(cloudlet)
21     else
22         break while-loop
23 return cloudletVmMap
```

Let M be the number of machines in computing setup, N be the number of SLA-based jobs submitted in Cloud, and n be the number of cloudlets scheduled by *Fill* scheduler of SLA-RALBA. The time complexities of Fill scheduler (i.e., Algorithm 2) and Spill scheduler (i.e., Algorithm 3) are $O(M^2.n)$ and $O(M.N–n)$, respectively. Therefore, the overall time complexity of SLA-RALBA is $O(M^2n)$, where M ≪ N on real Cloud.

Algorithm 3: SpillScheduler

Input: $cloudletSLAList$ - list of cloudlets $c_1, c_2, c_3, \ldots, c_n$,
$vmList$ - list of VMs $v_1, v_2, v_3, \ldots, v_m$,
$cloudletVmMap$ - set of cloudlets to VMs mapping by $FillScheduler$
Output: $cloudletVmMap$ - set of cloudlets to VMs mapping

```
1  vm = Null
2  cloudlet = Null
3  while cloudletSLAList.size() ≥ 1 do
4      cloudlet = getMaxCloudlet(cloudletSLAList)
5      if cloudlet is SLA-3 then
6          vm = getMostEconomicVm(cloudlet,vmList)
7      else if cloudlet is SLA-2 then
8          vm = getSuitableVM(cloudlet,vmList)
9      else if cloudlet is SLA-1 then
10         vm = getVMWithEFT(cloudlet,vmList)
11     cloudletVmMap.add(cloudlet,vm)
12     cloudletSLAList.remove(cloudlet)
13 return cloudletVmMap
```

4 Results and discussion

This section encompasses the experimental evaluation of SLA-RALBA in comparison with scheduling heuristics (i.e., Profit-MCT, Profit-Min-Min, Execution-Min-Min, Execution-MCT, SLA-MCT, and SLA-Min-Min) using two benchmark datasets, GoCJ [14] and HCSP instances [15, 16].

4.1 Experimental testbed and workload

The use of a real Cloud testbed for investigation of scheduling and allocation policies (with a varying heterogeneity, load and system size) restricts the experiments to the scale of the test cloud environment. Cloud computing is based on a pay-per-use model; thus, repeatable experiments on a real Cloud may incur a high monetary cost. Therefore, an ideal alternative is to use a simulation environment

Table 4 Configuration of the simulation environment

Simulator/version	CloudSim version 3.0.2
Total host machines	26 Host machines (Quad-core, each core with 4000 MIPS)
Host machine memory	16,384 MBs each
Total VMs	30 VMs (as shown in Fig. 3) 16 VMs (as shown in Figs. 4, 5)
Total cloudlets	200, 300, 400, 500, 600, 700, 800, 900, 1000 cloudlets in GoCJ dataset and 512 cloudlets in each instance of HCSP dataset

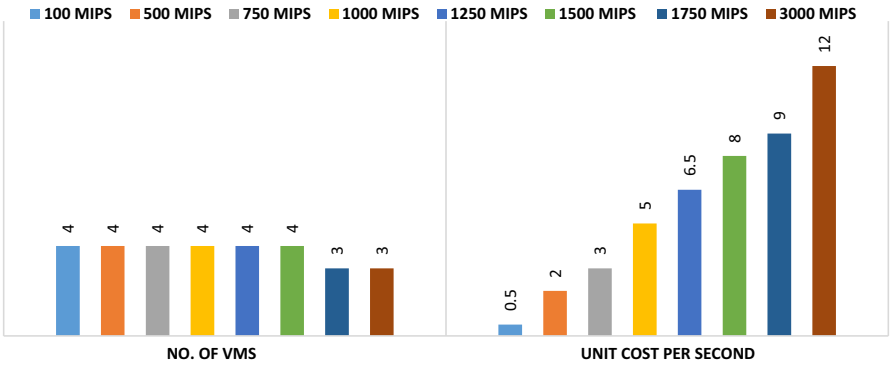
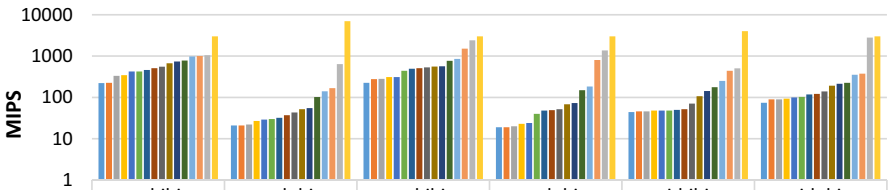


Fig. 3 Composition of 30 VMs in computing setup for GoCJ Dataset

that enables Cloud developers to conduct experiments by employing the desired and varying Cloud configurations. The performance evaluation of proposed SLA-RALBA and the other benchmark algorithms using different instances of benchmark datasets with a varying heterogeneity of computing environment is performed using a renowned Cloud simulator CloudSim (version 3.0.2) [17]. A novel feature of CloudSim is its support for modeling a large-scale Cloud computing environment on a single physical computing system with a little programming and developing effort. CloudSim provides a controlled and easy to set up a self-contained simulation platform for modeling Clouds, datacenters, provisioning, and allocation policies to test the performance of newly developed Cloud services.

In this work, the experiments are performed on a machine equipped with Intel Core i3-4030U Quad-core 1.9 GHz processor and 04GBs main memory. The configuration of the simulation environment is presented in Table 4. The simulation environment contains host machines in the Cloud datacenter (which virtualizes the computing resources in the form of VMs). All the experiments are conducted using 30 VMs hosted on 26 host machines within a datacenter. Figure 3 shows the information of the VMs (i.e., the computing power in terms of MIPS and execution cost in terms of unit cost per second). The computing powers of VMs vary from 100 to 3000 MIPS for the experimentation based on GoCJ dataset [14].



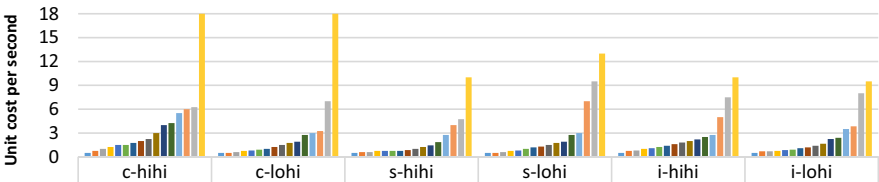
	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
VM-1	222	21	226	19	44	74
VM-2	226	21	277	19	46	90
VM-3	333	22	281	20	46	90
VM-4	345	27	310	23	48	93
VM-5	424	29	311	24	48	100
VM-6	424	30	441	40	48	103
VM-7	460	32	493	48	50	118
VM-8	511	37	509	49	52	122
VM-9	553	43	531	52	71	140
VM-10	669	52	559	68	107	192
VM-11	742	55	568	73	143	214
VM-12	781	102	766	149	177	226
VM-13	982	141	855	184	250	353
VM-14	1000	167	1516	805	441	375
VM-15	1046	641	2405	1375	504	2795
VM-16	3000	7000	3000	3000	4000	3000

Fig. 4 Computing powers of 16 VMs in computing setup for HCSP instances

A comprehensive analysis is performed to investigate the performance of the proposed SLA-RALBA heuristic in comparison with existing scheduling heuristics using two scientific benchmark datasets. One benchmark is *Expected Time to Compute* (ETC) model of Braun et al. [15, 16] in the form of HCSP instances and the other dataset is GoCJ dataset [14]. Instances in HCSP are categorized as with a name pattern "*c-thmh*," where *c* indicates the consistency type (*c* for consistent, *s* for semi-consistent, and *i* for inconsistent) and "*th*" and "*mh*" indicate the task and machine-level heterogeneity, respectively (*lo* and *hi* is used to represent low and high heterogeneity levels, respectively). The low and high task heterogeneity with high machine heterogeneity is employed for the experimental evaluation. Therefore, six HCSP instances employed in this pragmatic examination are *c-hihi, c-lohi, i-hihi, i-lohi, s-hihi,* and *s-lohi* instances. In addition, nine GoCJ instances are also used for experimentation; each instance is comprised of 200, 300, 400, 500, 600, 700, 800, 900, and 1000 cloudlets, respectively.

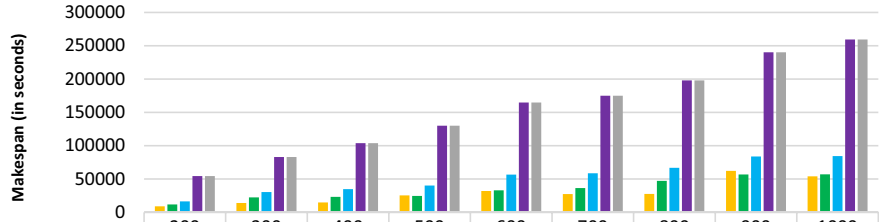
4.2 Simulation results: GoCJ dataset

The efficacy of the SLA-RALBA and existing scheduling techniques (i.e., Execution-MCT, Execution-Min-Min, Profit-MCT, Profit-Min-Min, SLA-MCT, and



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
VM-1	0.5	0.5	0.5	0.5	0.5	0.5
VM-2	0.75	0.5	0.6	0.5	0.75	0.7
VM-3	1	0.6	0.6	0.6	0.8	0.7
VM-4	1.25	0.75	0.75	0.75	1	0.75
VM-5	1.5	0.8	0.75	0.8	1.1	0.85
VM-6	1.5	0.9	0.75	1	1.25	0.9
VM-7	1.75	1	0.75	1.2	1.4	1.1
VM-8	2	1.25	0.85	1.3	1.6	1.2
VM-9	2.25	1.5	1	1.5	1.8	1.4
VM-10	3	1.75	1.25	1.75	2	1.65
VM-11	4	1.9	1.45	1.9	2.2	2.25
VM-12	4.25	2.75	1.85	2.75	2.5	2.4
VM-13	5.5	3	2.75	3	2.75	3.5
VM-14	6	3.25	4	7	5	3.85
VM-15	6.25	7	4.75	9.5	7.5	8
VM-16	18	18	10	13	10	9.5

Fig. 5 Expected gain of 16 VMs in computing setup for HSCP instances

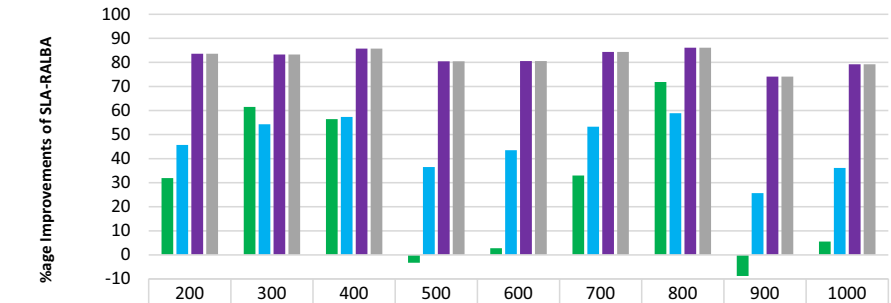


	200	300	400	500	600	700	800	900	1000
SLA-RALBA	8935	13874	14802	25446	31992	27384	27471	62180	53927
SLA-MCT	11787	22409	23154	24614	32883	36423	47202	56746	56937
SLA-Min-Min	16450	30340	34672	40068	56630	58582	66818	83678	84449
Profit-MCT	54439	83052	103739	130003	164682	174987	197916	240094	259324
Profit-Min-Min	54439	83052	103739	130003	164682	174987	197916	240094	259324

No. of Cloudlets

Fig. 6 Makespan results—GoCJ instances

SLA-Min-Min) is evaluated using *average resource utilization ratio* (ARUR), *Expected Gain* (in terms of execution cost), and makespan metrics. Each experiment is performed ten times, and the average values are reported for the results.

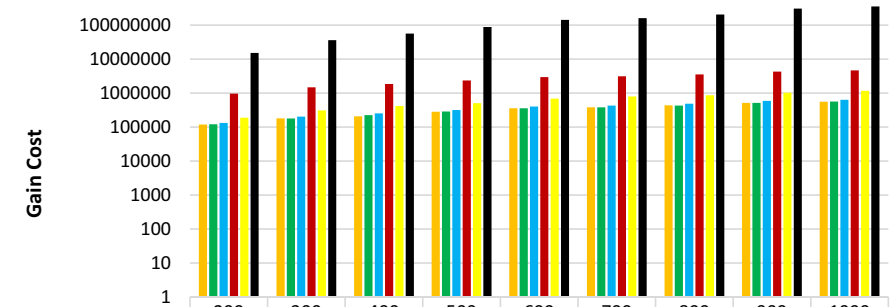


	200	300	400	500	600	700	800	900	1000
SLA-MCT	31.92	61.52	56.43	-3.27	2.79	33.01	71.83	-8.74	5.58
SLA-Min-Min	45.68	54.27	57.31	36.49	43.51	53.26	58.89	25.69	36.14
Profit-MCT	83.59	83.29	85.73	80.43	80.57	84.35	86.12	74.1	79.2
Profit-Min-Min	83.59	83.29	85.73	80.43	80.57	84.35	86.12	74.1	79.2

No. of Cloudlets

Fig. 7 %Age makespan improvements of SLA-RALBA—GoCJ Instances

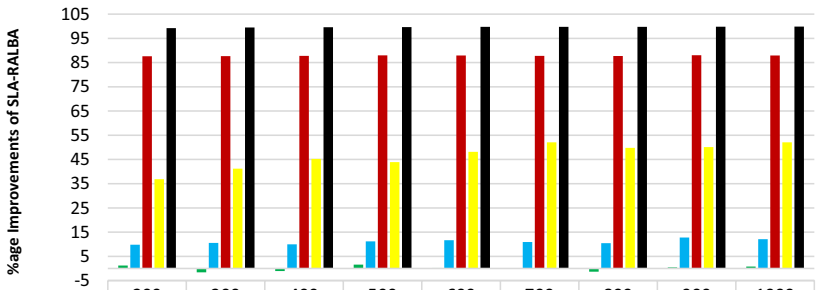
Figure 6 displays the makespan results for the instances of GoCJ dataset with 200, 300, 400, 500, 600, 700, 800, 900, and 1000 cloudlets. Figure 7 shows the percentage improvement in SLA-RALBA in terms of makespan as compared to SLA-MCT, SLA-Min-Min, Profit-MCT, and Profit-Min-Min heuristics for the given instances of GoCJ dataset. On average, the SLA-RALBA attains lower makespan than SLA-MCT (27.89% lower), SLA-Min-Min (45.69% lower), Profit-MCT (81.93% lower), and Profit-Min-Min (81.93% lower) heuristics.



	200	300	400	500	600	700	800	900	1000
SLA-RALBA	119060	180979	207478	282094	356245	381284	434775	513858	557906
SLA-MCT	120498	178164	225450	286597	357024	380503	429095	515987	562440
SLA-Min-Min	131978	202325	252974	317618	403392	427956	485569	589179	634482
Execution-MCT	960756	1470758	1860584	2342258	2951380	3115068	3538362	4289210	4629043
Execution-Min-Min	188477	307727	416028	503154	687299	796058	866247	1030159	1163987
RALBA	1.5E+07	3.6E+07	5.6E+07	8.8E+07	1.4E+08	1.6E+08	2.1E+08	3E+08	3.5E+08

No. of Cloudlets

Fig. 8 Gain cost results—GoCJ instances

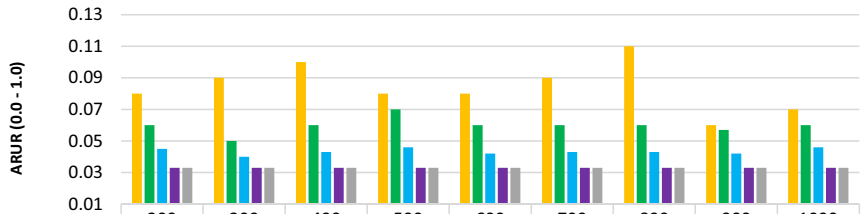


	200	300	400	500	600	700	800	900	1000
SLA-MCT	1.19	-1.58	-1.04	1.57	0.22	-0.21	-1.32	0.41	0.81
SLA-Min-Min	9.79	10.55	9.95	11.18	11.69	10.91	10.46	12.78	12.07
Execution-MCT	87.61	87.69	87.76	87.96	87.93	87.76	87.71	88.02	87.95
Execution-Min-Min	36.83	41.19	45.24	43.93	48.17	52.1	49.81	50.12	52.07
RALBA	99.21	99.49	99.59	99.67	99.75	99.76	99.78	99.83	99.84

No. of Cloudlets

Fig. 9 %Age gain cost improvements of SLA-RALBA—GoCJ instances

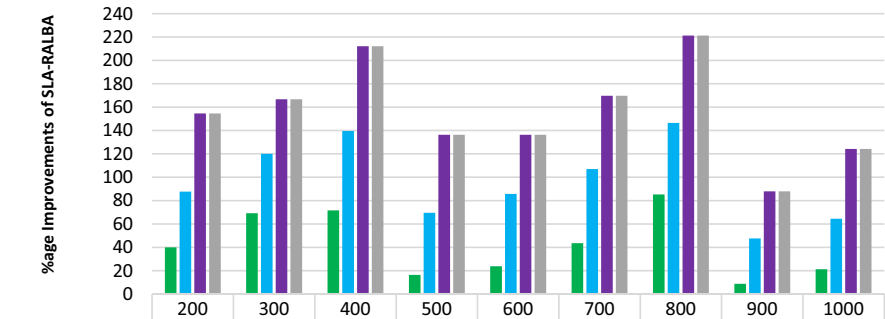
Figure 8 presents gain cost-based results of SLA-RALBA as compared to SLA-MCT, SLA-Min-Min, Execution-MCT, and Execution-Min-Min (for the instances of GoCJ dataset with 200, 300, 400, 500, 600, 700, 800, 900, and 1000 jobs). Figure 9 shows the improved percentage of SLA-RALBA in terms of gain cost against SLA-MCT, SLA-Min-Min, Execution-MCT, and Execution-Min-Min heuristics. On average, the SLA-RALBA achieves lower gain cost as compared to SLA-MCT (0.01% lower), SLA-Min-Min (11.04% lower), Execution-MCT (87.82% lower), and Execution-Min-Min (46.61% lower) heuristics. In addition, the cost efficiency of SLA-RALBA is also compared with RALBA scheduler [13], which is a non-SLA



	200	300	400	500	600	700	800	900	1000
SLA-RALBA	0.08	0.09	0.1	0.08	0.08	0.09	0.11	0.06	0.07
SLA-MCT	0.06	0.05	0.06	0.07	0.06	0.06	0.06	0.057	0.06
SLA-Min-Min	0.045	0.04	0.043	0.046	0.042	0.043	0.043	0.042	0.046
Profit-MCT	0.033	0.033	0.033	0.033	0.033	0.033	0.033	0.033	0.033
Profit-Min-Min	0.033	0.033	0.033	0.033	0.033	0.033	0.033	0.033	0.033

No. of Cloudlets

Fig. 10 ARUR results—GoCJ instances



	200	300	400	500	600	700	800	900	1000
SLA-MCT	40	69.23	71.67	16.42	23.81	43.55	85.31	8.77	21.31
SLA-Min-Min	87.67	120	139.53	69.57	85.71	106.98	146.51	47.62	64.44
Profit-MCT	154.55	166.67	212.12	136.36	136.36	169.7	221.21	87.88	124.24
Profit-Min-Min	154.55	166.67	212.12	136.36	136.36	169.7	221.21	87.88	124.24

No. of Cloudlets

Fig. 11 %Age ARUR improvements of SLA-RALBA—GoCJ instances

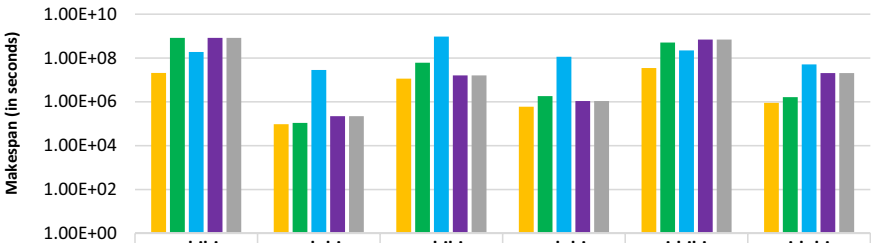
load balancing algorithm for Cloud computing. On average, SLA-RALBA attains 99.66% lower gain cost as compared to RALBA technique as shown in Fig. 9.

Figure 10 shows the comparison of ARUR-based results of SLA-RALBA with SLA-MCT, SLA-Min-Min, and Profit-MCT, and Profit-Min-Min for the instances of GoCJ dataset with 200, 300, 400, 500, 600, 700, 800, 900, and 1000 jobs. Figure 11 shows the percentage improvement in SLA-RALBA in terms of ARUR against SLA-MCT, SLA-Min-Min, Profit-MCT, and Profit-Min-Min heuristics. On average, the SLA-RALBA attains higher resource utilization as compared to SLA-MCT (42.23% higher), SLA-Min-Min (96.34% higher), Profit-MCT (156.57% higher), and Profit-Min-Min (156.57% higher) heuristics. On the other hand, the Execution-MCT and Execution-Min-Min attain the highest resource utilization as compared to SLA-RALBA. In contrary, the Execution-MCT and Execution-Min-Min attain 1965% and 982% higher gain as compared to SLA-RALBA, because these two heuristics do not take into account the SLA parameters in scheduling decisions.

4.3 Simulation results: HCSP dataset [15, 16]

Figure 12 presents the makespan-based results for the six instances of 512×16 HCSP dataset, namely *u-c-hihi, u-c-lohi, u-s-hihi, u-s-lohi, u-i-hihi*, and *u-i*-lohi. Figure 13 shows the improved percentage of SLA-RALBA in terms of makespan against SLA-MCT, SLA-Min-Min, Profit-MCT, and Profit-Min-Min heuristics for the given instances of HCSP dataset. On average, the SLA-RALBA attains lower makespan than SLA-MCT (up to 66.17% lower), SLA-Min-Min (up to 94.91% lower), Profit-MCT (up to 69.94% lower), and Profit-Min-Min (up to 69.94% lower) heuristics.

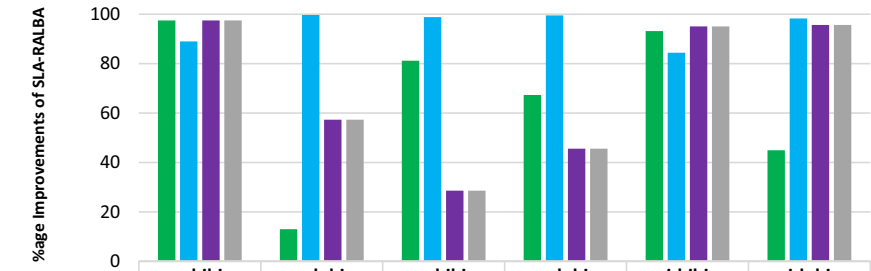
Figure 14 presents the comparison of gain cost-based results of SLA-RALBA with SLA-MCT, SLA-Min-Min, Execution-MCT, and Execution-Min-Min for the



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
SLA-RALBA	20932092	95604	11525698	596412	34860934	906629
SLA-MCT	827623549	109930	61235856	1825123	508278837	1647532
SLA-Min-Min	189329860	28492553	952750146	114853473	223080547	51165039
Profit-MCT	827623549	223911	16148630	1096182	700543972	20621604
Profit-Min-Min	827623549	223911	16148630	1096182	700543972	20621604

No. of Cloudlets

Fig. 12 Makespan results—HCSP instances

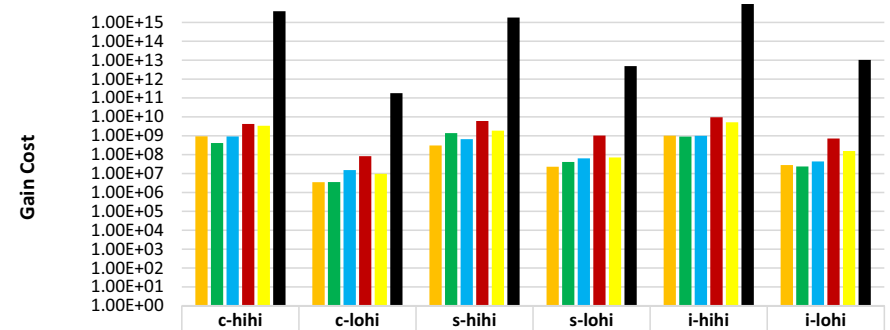


	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
SLA-MCT	97.47	13.03	81.18	67.32	93.14	44.97
SLA-Min-Min	88.94	99.66	98.79	99.48	84.37	98.23
Profit-MCT	97.47	57.3	28.63	45.59	95.02	95.6
Profit-Min-Min	97.47	57.3	28.63	45.59	95.02	95.6

No. of Cloudlets

Fig. 13 %Age makespan improvements of SLA-RALBA—HCSP instances

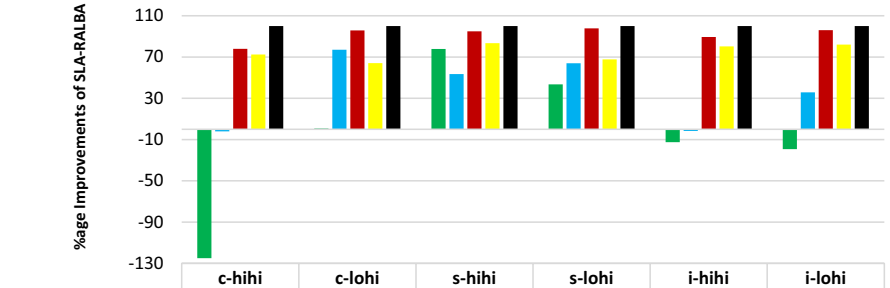
given instances of 512 × 16 HCSP dataset. Likewise, Fig. 15 shows the percentage of improvements in SLA-RALBA in terms of gain cost against SLA-MCT, SLA-Min-Min, Execution-MCT, and Execution-Min-Min heuristics. On average, the SLA-RALBA achieves lower gain as compared to SLA-Min-Min (i.e., 37.74% lower), Execution-MCT (i.e., 91.97% lower), and Execution-Min-Min (i.e., 75.02% lower) heuristics, respectively. SLA-RALBA is also compared with RALBA technique



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
SLA-RALBA	931471491	3501913	306383656	23094686	1013117399	28176771
SLA-MCT	413811775	3527427	1384644313	40861803	900371550	23627287
SLA-Min-Min	912434650	15275373	659362697	64101366	996119408	43826897
Execution-MCT	4212010808	83546680	6023862521	1024376709	9580794620	713815578
Execution-Min-Min	3388829450	9745480	1857322142	71460100	5139566514	156856822
RALBA	3.93217E+15	1.79987E+11	1.80605E+15	4.8762E+12	9.41938E+15	1.03635E+13

No. of Cloudlets

Fig. 14 Gain cost results—HCSP Instances



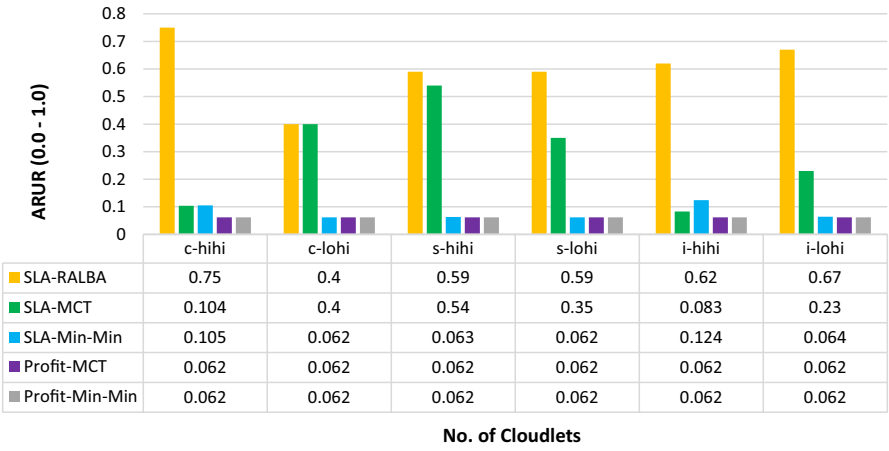
	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
SLA-MCT	-125	0.72	77.87	43.48	-12.52	-19.26
SLA-Min-Min	-2.09	77.07	53.53	63.97	-1.71	35.71
Execution-MCT	77.89	95.81	94.91	97.75	89.43	96.05
Execution-Min-Min	72.51	64.07	83.5	67.68	80.29	82.04
RALBA	100	99.99	100	99.99	100	99.99

No. of Cloudlets

Fig. 15 %Age gain cost improvements of SLA-RALBA—HCSP instances

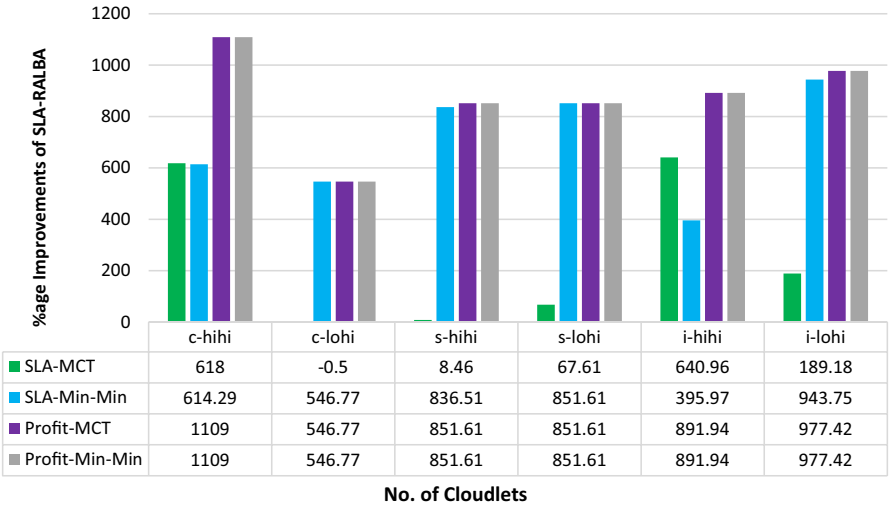
using HCSP dataset, and SLA-RALBA attains 99.9% lower gain as compared to RALBA scheduler as shown in Fig. 15. However, SLA-MCT achieves 5.79% lower gain on average (due to much better performance of SLA-MCT in terms of gain cost only using u-c-hihi instance of 512×16 HCSP dataset) as compared to the proposed SLA-RALBA.

Figure 16 presents the comparison of ARUR-based results of SLA-RALBA with SLA-MCT, SLA-Min-Min, and Profit-MCT and Profit-Min-Min for given instances



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
SLA-RALBA	0.75	0.4	0.59	0.59	0.62	0.67
SLA-MCT	0.104	0.4	0.54	0.35	0.083	0.23
SLA-Min-Min	0.105	0.062	0.063	0.062	0.124	0.064
Profit-MCT	0.062	0.062	0.062	0.062	0.062	0.062
Profit-Min-Min	0.062	0.062	0.062	0.062	0.062	0.062

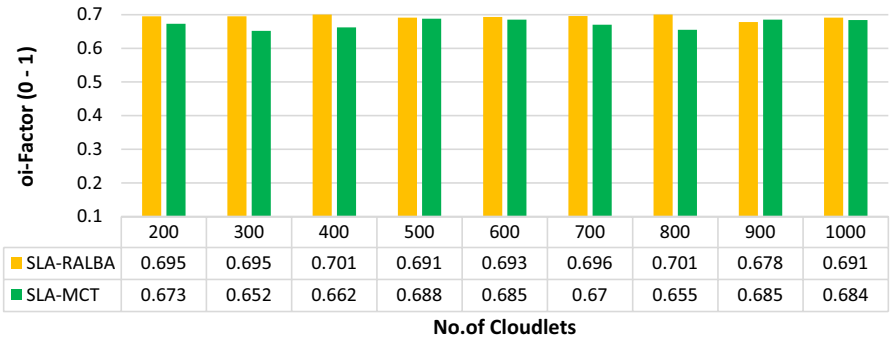
Fig. 16 ARUR results—HCSP instances



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
SLA-MCT	618	-0.5	8.46	67.61	640.96	189.18
SLA-Min-Min	614.29	546.77	836.51	851.61	395.97	943.75
Profit-MCT	1109	546.77	851.61	851.61	891.94	977.42
Profit-Min-Min	1109	546.77	851.61	851.61	891.94	977.42

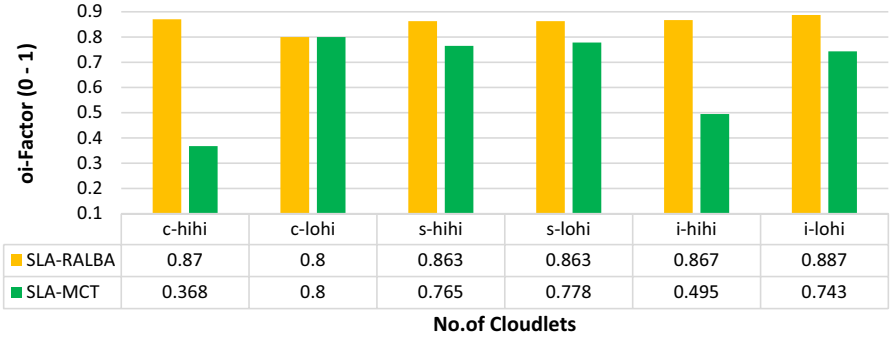
Fig. 17 %Age ARUR improvements of SLA-RALBA—HCSP instances

of 512×16 HCSP dataset. Figure 17 shows the percentage improvement in SLA-RALBA in terms of ARUR against SLA-MCT, SLA-Min-Min, Profit-MCT, and Profit-Min-Min heuristics. On average, the SLA-RALBA attains higher resource utilization as compared to SLA-MCT (253.95% higher), SLA-Min-Min (698.15% higher), Profit-MCT (871.39% higher), and Profit-Min-Min (871.39% higher) heuristics. On the other hand, the Execution-MCT and Execution-Min-Min heuristics attain 35.23% and 4.54% highest resource utilization, respectively, on average as compared to SLA-RABA. On the contrary, the Execution-MCT and



	200	300	400	500	600	700	800	900	1000
SLA-RALBA	0.695	0.695	0.701	0.691	0.693	0.696	0.701	0.678	0.691
SLA-MCT	0.673	0.652	0.662	0.688	0.685	0.67	0.655	0.685	0.684

Fig. 18 Overall improvement in SLA-RALBA against SLA-MCT—GoCJ instances



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
SLA-RALBA	0.87	0.8	0.863	0.863	0.867	0.887
SLA-MCT	0.368	0.8	0.765	0.778	0.495	0.743

Fig. 19 Overall improvement in SLA-RALBA against SLA-MCT—HCSP instances

Execution-Min-Min heuristics attain higher gain cost as compared to SLA-RALBA (as shown in Fig. 15).

4.4 Results and discussion

A substantial improvement is witnessed in the efficacy of SLA-RALBA in terms of resource utilization against existing SLA-aware and SLA-spare scheduling algorithms (as shown in Figs. 10, 11, 16, 17). Similarly, a significant enhancement in the makespan results is also achieved except against the makespan-based results of SLA-MCT for a couple of GoCJ instances (i.e., 500 and 900 cloudlets instances, as shown in Fig. 7). The SLA-RALBA attained a significant reduction in expected gain against existing algorithms except SLA-MCT for few GoCJ and HCSP instances (i.e., in case of c-hihi, i-hihi, and i-lohi instances of HCSP, and GoCJ instances with 300, 400, and 800 cloudlets, respectively). However, SLA-RALBA outperformed the given scheduling heuristics (including SLA-MCT) by offering a well-balanced among makespan, expected gain, and resource utilization on Cloud using all the employed instances of HCSP and GoCJ datasets.

To demonstrate the improved performance of SLA-RALBA against the SLA-MCT, an *Overall Improvement Factor* (oi-Factor) is calculated and compared as follows. The max–min normalization is adopted to normalize the values of makespan and expected gain (achieved by SLA-RALBA and SLA-MCT in the experimental results) to a number range of 0–1 [32, 33]. The normalized values of makespan and expected gain are represented as *n-Makespan* and *n-EGain*, respectively. The *n-Makespan* and *n-EGain* values vary between 0 and 1; a higher value of *n-Makespan* and *n-EGain* depicts better obtained results. However, the higher value of ARUR indicates that the scheduling algorithm has better inherent load-balanced execution. Therefore, *n-Makespan'* and *n-EGain'* are computed by subtracting the values of *n-Makespan* and *n-EGain* from 1, respectively. The *oi-Factor* of an algorithm is calculated by averaging its values of *n-Makespan', n-EGain',* and *ARUR*. Figures 18 and 19 show the overall improvement in SLA-RALBA against SLA-MCT using GoCJ and HCSP datasets, which proves a promising balance of SLA-RALBA among makespan, expected gain, and resource utilization.

5 Conclusions

A comprehensive investigation of the state-of-the-art SLA-based heuristics fallouts in reduced resource utilization and increased execution cost due to the improper allocation of jobs to VM and straggling jobs in scheduling. The proposed novel technique SLA-RALBA ensures enhanced resource utilization with reduced execution time and cost. The performance of SLA-RALBA has been compared using benchmark HCSP and GoCJ datasets with the scheduling algorithms in terms of execution time, cost, and resource utilization. The exhaustive examination of pragmatic results has shown that SLA-RALBA has presented an improved balance among the resource utilization, execution time, and cost than existing state-of-the-art algorithms (i.e., Execution-MCT, Execution-Min-Min, Profit-MCT, Profit-Min-Min, SLA-MCT, and SLA-Min-Min). In the future, we intend to consider other important QoS aspects to enhance the functionalities of SLA-RALBA, especially the communication cost involved in the scheduling on Cloud.

References

1. Beloglazov A, Buyya R (2013) Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. IEEE Trans Parallel Distrib Syst 24(7):1366–1379
2. Yeo CS, Buyya R (2005) Service level agreement based allocation of cluster resources : handling penalty to enhance utility. In: Proceedings of the 7th IEEE International Conference on Cluster Computing
3. Durao F, Fernando J, Carvalho S, Fonseka A (2014) A systematic review on cloud computing. J Supercomput 68(3):1321–1346
4. Groot S (2013) Research on efficient resource utilization in data intensive distributed systems. The University of Tokyo, Tokyo

5. Yadwadkar NJ, Gonzalez JE, Katz R (2016) Multi-task learning for straggler avoiding predictive job scheduling. J Mach Learn Res 17:1–37
6. Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I (2013) Effective straggler mitigation : attack of the clones. In: 10th USENIX Symposium on Networked Systems Design and Implementation, 2013, pp 185–198
7. Hussain A, Aleem M, Islam MA, Iqbal MA (2018) A rigorous evaluation of state-of-the-art scheduling algorithms for cloud computing. IEEE Access 6:1–15
8. Kavulya S, Tany J, Gandhi R, Narasimhan P (2010) An analysis of traces from a production MapReduce cluster. In: 11th IEEE/ACM International Conference on Grid Computing (CCGrid), 2010, pp 94–103
9. Isard M, Birrell A, Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. In: 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, 2007, pp 59–72
10. Zaharia M et al (2012) Resilient distributed datasets : a fault-tolerant abstraction for in-memory cluster computing. In: 9th USENIX Conference on Networked Systems Design and Implementation
11. Son S, Jung G, Chan S (2013) An SLA-based cloud computing that facilitates resource allocation in the distributed data centers of a cloud provider An SLA-based cloud computing that facilitates resource allocation in the distributed data centers of a cloud. J Supercomput 64(2):606–637
12. Garg SK, Toosi AN, Gopalaiyengar SK, Buyya R (2014) SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter. J Netw Comput Appl 45:108–120
13. Hussain A, Aleem M, Khan A, Iqbal MA, Islam MA (2018) RALBA: a computation-aware load balancing scheduler for cloud computing. Clust Comput 21(3):1667–1680. https://doi.org/10.1007/s10586-018-2414-6
14. Hussain A, Aleem M (2018) GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructure. MDPI Data 3(4):1–12
15. 'Heterogeneous computing scheduling problem (HCSP) instances'. [Online]. Available: https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/HCSP_inst.htm. Accessed 26 Feb 2019
16. Braun TD et al (2001) A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Distrib Comput 61(6):810–837
17. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R (2009) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp 39(7):701–736
18. Leitner P, Hummer W, Satzger B, Inzinger C, Dustdar S (2012) Cost-efficient and application SLA-aware client side request scheduling in an infrastructure-as-a-service cloud. In: IEEE Fifth International Conference on Cloud Computing, 2012, pp 213–220
19. Alrokayan M, Dastjerdi AV, Buyya R (2014) SLA-aware provisioning and scheduling of cloud resources for big data analytics. In: IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), 2014, pp 1–8
20. Sharma U, Shenoy P, Sahu S, Shaikh A (2011) A cost-aware elasticity provisioning system for the cloud. In: IEEE 31st International Conference on Distributed Computing Systems, 2011, pp 559–570
21. Lenzini L, Mingozzi E, Stea G (2004) Tradeoffs between low complexity, low latency, and fairness with deficit round-robin schedulers. IEEE/ACM Trans Netw 12(4):681–693
22. Aditya A, Chatterjee U, Gupta S (2015) A comparative study of different static and dynamic load balancing algorithm in cloud computing with special emphasis on time factor. Int J Curr Eng Technol 5(3):2277–4106
23. Hamid S et al (2017) Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment. PLoS ONE 12(5):1–26
24. Li B, Wu H (2015) Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds. J Supercomput 71(8):3009–3036
25. Muhammed A, Abdullah A, Hussin M (2016) Max-average: an extended max–min scheduling algorithm for grid computing environment. J Telecommun Electron Comput Eng 8(6):43–47
26. Elzeki OM, Rashad MZ, Elsoud MA (2012) Overview of scheduling tasks in distributed computing systems. Int J Soft Comput Eng 2(3):470–475
27. Tchernykh A et al (2016) Online Bi-objective scheduling for IaaS clouds ensuring quality of service. J Grid Comput 14(1):5–22
28. Maheswaran M, Ali S, Siegel HJ, Hensgen D, Freund RF (1999) Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J Parallel Distrib Comput 59(2):107–131

29. Chen H, Wang F, Helian N, Akanmu G (2013) User-priority guided min-min scheduling algorithm for load balancing in cloud computing. In: 2013 National Conference on Parallel Computing Technologies, PARCOMPTECH 2013, pp 1–8
30. Sharma G, Banga P (2013) Task aware switcher scheduling for batch mode mapping in computational grid environment. Int J Adv Res Comput Sci Softw Eng 3:1292–1299
31. Panda SK, Jana PK (2017) SLA-based task scheduling algorithms for heterogeneous multi-cloud environment. J Supercomput 73(6):2730–2762
32. Al Shalabi L, Shaaban Z, Kasasbeh B (2006) Data mining: a preprocessing engine data mining. J Comput Sci 2(9):735–739
33. Shao X, Wang Z, Li P, Feng CJ (2005) Integrating data mining and rough set for customer group-based discovery of product configuration rules. Int J Prod Res 44(14):2789–2811

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.