Cluster Computing (2018) 21:1667–1680
https://doi.org/10.1007/s10586-018-2414-6

# RALBA: a computation-aware load balancing scheduler for cloud computing

Altaf Hussain[1] · Muhammad Aleem[1] · Abid Khan[2] · Muhammad Azhar Iqbal[1] · Muhammad Arshad Islam[1]

Received: 23 August 2017 / Revised: 21 January 2018 / Accepted: 6 March 2018 / Published online: 14 March 2018


**Abstract**
Cloud computing serves as a platform for remote users to utilize the heterogeneous resources in data-centers to compute *High-Performance Computing* jobs. The physical resources are virtualized in Cloud to entertain user services employing *Virtual Machines* (VMs). Job scheduling is deemed as a quintessential part of Cloud and efficient utilization of VMs by *Cloud Service Providers* demands an optimal job scheduling heuristic. An ideal scheduling heuristic should be efficient, fair, and starvation-free to produce a reduced makespan with improved resource utilization. However, static heuristics often lead to inefficient and poor resource utilization in the Cloud. An idle and underutilized host machine in Cloud still consumes up to 70% of the energy required by an active machine (Ray, in Indian J Comput Sci Eng 1(4):333–339, 2012). Consequently, it demands a load-balanced distribution of workload to achieve optimal resource utilization in Cloud. Existing Cloud scheduling heuristics such as Min–Min, Max–Min, and Sufferage distribute workloads among VMs based on minimum job completion time that ultimately causes a load imbalance. In this paper, a novel *Resource-Aware Load Balancing Algorithm* (RALBA) is presented to ensure a balanced distribution of workload based on computation capabilities of VMs. The RABLA framework comprises of two phases: (1) scheduling based on computing capabilities of VMs, and (2) the VM with earliest finish time is selected for jobs mapping. The outcomes of the RALBA have revealed that it provides substantial improvement against traditional heuristics regarding makespan, resource utilization, and throughput.

**Keywords** Cloud scheduling · Load balancing · Computation-aware scheduling · Resource utilization · Cloud simulation

## 1 Introduction

Cloud computing [1] is a business-oriented concept to outsource the elastic and scalable IT resources as a utility computing [2]. One of the fundamental aspects of this paradigm is the guaranteed on-demand availability of distributed resources over the Internet to solve *High-Performance Computing* (HPC) problems [3]. In general, scheduling is significant and challenging issue known as an NP-complete problem [4–8] and particularly it must be addressed to achieve good performance in HPC environment. The nature of scheduling strategies could be either *static* or *dynamic*. The dynamic strategies can be classified into two types: *online* and *batch* [9]. Online dynamic strategies deal with the scheduling of a single job, whereas the batch dynamic scheduling strategies are concerned with the mapping of job batch. In general, the low scheduling overhead of static strategies adheres them to outperform over dynamic strategies [10–12]. In addition, static scheduling heuristics avoid *Virtual Machine* (VM) migration to reduce the execution delays and provide irrefutable *Quality of Service* (QoS) [13]. On the other hand, static heuristics usually produce inefficient and poor resource utilization due to fixed resource allocation [10].

Improper workload distribution on VMs causes longer execution time and more energy consumption due to which most of the scheduling heuristics are not efficient enough to harness the full capacity of the available Cloud computing resources. According to a survey [1], approximately 1.6 million tons of additional $CO_2$ emission are caused only due to the idle computing resources within Cloud data-

✉ Muhammad Aleem
aleem@cust.edu.pk

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

[2] Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, Pakistan

centers. Moreover, the idle computing resources cost additional 19 billion dollars in terms of the consumed resources and energy cost. To overcome the issue of load imbalance, several researchers have proposed scheduling mechanisms [14–16] to maps Cloud jobs in a load balanced manner to achieve improved resource utilization and system throughput [17]. Min–Min scheduling heuristic [18] offers a reduced makespan for a job pool comprised of small sized jobs. On the other hand, if the job pool contains very large size jobs the Min–Min heuristic produces load imbalance, low resource utilization, and high makespan. On the other hand, Max–Min [19] provides a quality solution with reduced makespan for a job pool with a large number of shorter jobs and few longer ones. However, the inherent mechanism of Min–Min and Max–Min provides desired makespan but often leads to a poor resource utilization [17–19]. Most of the proposed scheduling mechanisms do not consider mapping of Cloud jobs in a load balanced manner, which ultimately results in poor resource utilization and low system throughput.

This study proposes a scheduling algorithm named *Resource-Aware Load Balancing Algorithm* (RALBA) to mitigate load imbalance issue in Cloud computing. RALBA is a novel batch-dynamic scheduling heuristic wherein Cloud jobs (in batch) are scheduled in a load balanced manner. RALBA schedules a batch of compute-intensive, non-preemptive, and independent jobs. The primary objective of RALBA is to maximize resource utilization, minimize execution time or makespan, and maximize throughput. RALBA performs execution in two phases; in the first phase, the workload is scheduled according to the computing capabilities of *Virtual Machines* (VMs) and computing requirements of Cloud jobs. The second phase schedules the remaining jobs (left by the first phase scheduling of RALBA) to VMs producing the *Earliest Finish Time* (EFT). Experimental evaluation of RALBA has revealed that the proposed scheduling heuristic has achieved up to 7.21 times improved resource utilization as compared to *Random Selection* (RS) heuristic. The main contributions of the proposed scheme are as follows:

- In-depth analysis of the current state-of-the-art to identify merits and demerits of several existing heuristics;
- A novel load-balancing Cloud scheduler for a batch of compute-intensive, non-preemptive, and independent jobs that produces reduced makespan, increased resource utilization, and higher throughput, and;
- Performance analysis and empirical investigation of the proposed scheduling heuristic against state-of-the-art scheduling techniques.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents the system architecture, system model, and RALBA algorithm along with the complexity and overhead analysis. The experimental setup, workload characteristics, performance evaluation, and discussion related to the attained performance results are presented in Sect. 4. In the end, Sect. 5 concludes the paper and identifies some potential future directions of this work.

## 2 Related work

*Random Selection* (RS) Cloud scheduling allocates jobs to VMs without considering the current load of VMs [20–22]. RS has a minimal scheduling overhead, simple implementation, and low complexity compared to other Cloud scheduling heuristics [23]. However, RS arbitrarily assigns a job to a randomly selected VM. However, the unfair scheduling mechanism employed by the RS may lead to the selection of an overloaded VM that could cause the load-imbalance, low resource utilization, and a long waiting time for the submitted jobs [20, 21].

*Round Robin* (RR) scheduling mechanism distributes Cloud jobs over the available VMs in a circular order [21, 23]. The equitable scheduling of RR results in a mapping of an almost equal number of jobs to VMs regardless of the job sizes [21, 23]. RR has minimal scheduling overhead as compared to the other scheduling techniques [20, 21, 24]. However, the assignment of small jobs to the faster resources and large jobs to the slower resources originates the longer makespan, poor resource utilization, and load imbalance [20, 24].

*Minimum Completion Time* (MCT) heuristic assigns the candidate job to a VM producing minimum completion time for it [7, 22, 25, 26]. The current load of a VM is employed to identify appropriate VM for assignment of the job [7, 20]. At each scheduling step, MCT heuristic has to scan all the available VMs to find the machine producing minimum completion time for a candidate job that causes significant scheduling overhead. On the other hand, MCT heuristic produces improved makespan and resource utilization compared to the RR and RS techniques [27]. Another concern of the MCT is that it assigns more jobs to faster VMs that leads to load imbalance [17, 24, 25].

Min–Min heuristic is based on the MCT mechanism [7, 26–28]. The functionality of this heuristic follows two steps; firstly, the earliest finish time of all jobs is determined by considering all VMs. In the second step, the job with the minimum earliest finish time is selected and assigned to the concerned VM. On each scheduling decision, the ready time of the candidate VM is updated and this process continues until all the jobs have been scheduled on the VMs. Min–Min heuristic favors smaller jobs and penalizes larger jobs [17, 20, 29]. Min–Min often results in low resource utilization for a job pool based on fewer larger jobs

[21, 25, 27]. Moreover, Min–Min overloads faster VMs with smaller jobs that lead to load imbalance.

Max–Min has a resemblance to Min–Min. Both heuristics have a divergent job selection policy but the functionality of both the heuristics is almost identical. In Max–Min, the first step is identical to Min–Min but in the second step, the job with the maximum earliest finish time is selected and assigned to the concerned VM [26, 27, 30]. On each scheduling decision, the ready time of the VM is updated and the process is repeated until all the jobs have been scheduled. Max–Min favors larger jobs by producing minimum completion time; on the other hand, Max–Min penalizes smaller size jobs [18, 22, 29–31]. Moreover, Max–Min heuristic produces load imbalance for a job pool with larger size jobs [8, 18, 31].

*Resource-Aware Scheduling Algorithm* (RASA) [18, 29] uses Min–Min and Max–Min alternatively to utilize the merits of both heuristics. RASA was originally proposed for grid computing. First, RASA develops a job-related resource-matrix that contains the completion time of each job on all the resources. For the mapping of a first job, Min–Min heuristic is employed if numbers of jobs are odd; otherwise, Max–Min is used for mapping the jobs. After that, all the remaining jobs are scheduled using one of the two algorithms (i.e., Min–Min and Max–Min) alternatively. RASA provides fair scheduling for both large and small size jobs [18, 23, 28]. On the other hand, RASA generates load imbalance [21] and penalizes smaller jobs if the workload contains a large number of big jobs [30, 32].

Sufferage heuristic [16, 23, 24] computes the sufferage-value for each job by finding the difference between its MCT and the second MCT (higher than the first MCT) generated by any VM. The job with largest *sufferage-value* is assigned to the VM producing minimum completion time for it. Sufferage produces minimal makespan; but it has a consequential scheduling overhead due to the large number of calculations (to compute sufferage-value) in each scheduling decision [31]. Mostly, Sufferage produces minimal makespan compared to RS, RR, MCT, Min–Min, and Max–Min [23, 28].

*Task-Aware Scheduling Algorithm* (TASA) favors smaller jobs in one step by using Min–Min and finds an appropriate VM for the job by using Sufferage in the second step [33]. TASA generates better makespan and resource utilization as compared to Min–Min, Max–Min, RASA, and Sufferage [33].

Panda et al. [16] proposed a *Skewness-Based Min–Min Max–Min* ($SBM^2$) for scheduling of *skewed* size workload on Grid. If the dataset contains a large number of shorter jobs with a few longer jobs then the dataset is referred as *positively skewed*. On the other hand, if the dataset is comprised of a large number of longer jobs with a few shorter jobs then the dataset is known as *negatively skewed* [16]. $SBM^2$ computes the skewness of the workload in each scheduling decision. Min–Min is employed if the workload is negatively skewed; otherwise, Max–Min is adopted for the scheduling of positively skewed workload.

*Priority-Aware Load-Balanced Improved Min–Min* (PA-LBIMM) heuristic is proposed in [15]. PA-LBIMM categorizes both the jobs and resources in *VIP* and *normal* classes. First, the *VIP* jobs are scheduled on *VIP* resources using Min–Min. After that, PA-LBIMM schedules the *normal* jobs on all resources using Min–Min.

*Service Level Agreement Min–Min* (SLA-Min–Min) and *Service Level Agreement MCT* (SLA-MCT) are proposed in [34]. SLA-Min–Min and SLA-MCT are SLA-based heuristics, which provide a quality solution to form a balance between makespan and penalty cost on execution time. Table 1 presents a concrete summary of the related work.

Synthesis of the literature has revealed that most of the contemporary scheduling heuristics often result in the form of low resource utilization and load imbalance. The empirical analysis (in our experiments) exhibits that a smaller makespan produced by the existing scheduling techniques cannot guarantee the load balanced mapping of the jobs.

## 3 Proposed load balancing algorithm

This section presents a detailed overview of the proposed scheduling heuristic RALBA. The system architecture, overall system and performance model, algorithm, computational complexity, and overhead analysis are presented in this section.

### 3.1 RALBA overview and background

RALBA is a resource-aware load balancing algorithm comprises of two sub-schedulers: *Fill* and *Spill*. Figure 1 presents the abstraction layer of RALBA based Cloud computing architecture. A renowned simulator CloudSim [35] is employed to evaluate the performance of the proposed RALBA scheduler. *Cloudlet* is used as a synonym for the job in CloudSim simulator. Physical and virtual instance layers provide a foundation for the delivery of *Infrastructure-as-a-Service* (IaaS) and *Platform-as-a-Service* (PaaS) to the Cloud users [35]. The computing power of a Cloud data-center is represented as a collection of physical host machines along with the storage servers at the physical instance layer. These resources are transparently managed by harnessing the virtualization concept to provide dynamic sharing of computing and storage resources at the virtual instance layer. A Cloud resource manager maintains the track and records the status of VMs in the virtual instance layer. The resource manager is accountable for creation, termination, and migration of VMs when

**Table 1** Summary of the related work

| Heuristics | Strengths | Weaknesses |
|---|---|---|
| RS [20–22] | Minimal scheduling overhead [24], simple implementation with low complexity [23], | No fairness in scheduling [20, 21], produces load-imbalance [20, 21] |
| RR [21, 23] | Fairness in scheduling [21, 24], minimal scheduling overhead [23, 25] | Generally poor makespan [22], poor resource utilization [24], load-imbalance [20, 24] |
| MCT [9, 11, 12] | Improved makespan than RS and RR [27], machine-aware scheduling [20] | Faster resource overloaded with more jobs [22, 23], load-imbalance [24, 25, 28], |
| Min–Min [25–27] | Favors smaller jobs [29], reduced makespan for smaller jobs [17, 20] | Penalize larger jobs [22], load imbalance [17, 23, 26, 31], overloads faster machines with smaller jobs [17, 20, 29, 31] |
| Max–Min [16, 30, 36, 45] | Favors larger jobs [20, 36], reduced makespan for larger jobs [24, 29, 33] | Penalize smaller jobs [29, 46]., load-imbalance for job pool with more larger jobs [7, 20, 24, 26, 29] |
| Sufferage [18, 25, 26] | Improved makespan than RS, RR, MCT, Min–Min, and Max–Min [26, 28], job allocation to appropriate machine [25, 26] | More scheduling overhead due to computing sufferage-value in each scheduling decision [31] |
| TASA [33] | Improved makespan than Max–Min, Min–Min, RASA, and Sufferage [33], favor smaller jobs [33] | Load balancing is not considered [33] |
| RASA [21, 29] | Fair treatment of larger and smaller jobs [18, 23, 28], relatively improved makespan than Min–Min and Max–Min [18, 26] | Load-imbalance [21], penalize smaller jobs in some cases [32] |
| $SBM^2$ [16] | Fair treatment for positively and negatively skewed workload [16], reduced makespan than Min–Min and Max–Min [16] | Heap of calculation in finding positive and negative skewness in each scheduling decision [16], Load balancing not considered |
| PA-LBIMM [15] | Improved makespan for *VIP* jobs [15], priority-aware scheduling | Penalize normal jobs due to *VIP* jobs, relatively increased makespan than Min–Min [15] |

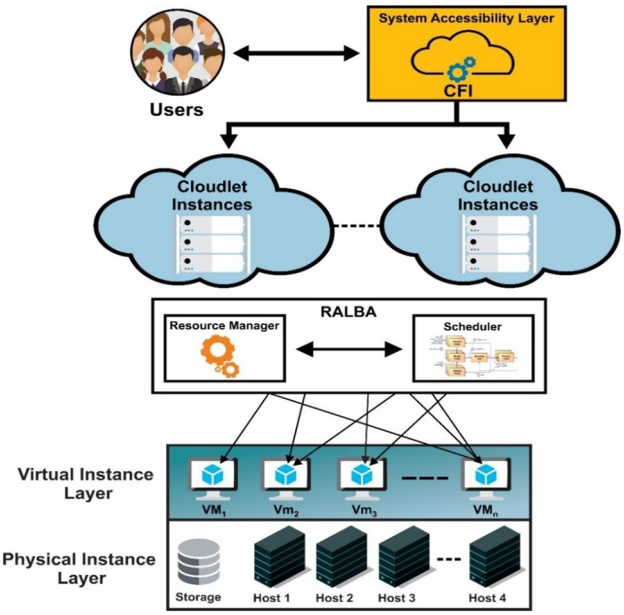



**Fig. 1** Abstraction layer of RALBA architecture

required. Cloud resource manager provides to RALBA the information pertaining to the available VMs and their computing capabilities. To utilize the VMs for their full capacity, a resource-aware scheduler is required on top of the virtual instance layer for a balanced distribution of cloudlets among VMs. The system accessibility layer provides a user-friendly interface for the submission of HPC based cloudlet instances to the Cloud. To improve resource utilization and balanced job scheduling, we propose RALBA on top of the virtual instance layer (as shown in Fig. 1).

RALBA system and performance model are presented using mathematical expressions. The basic definitions, terminologies, and the notations used in the mathematical expressions are listed in Table 2.

### 3.2 RALBA system architecture

RALBA is based on batch dynamic scheduling technique wherein a batch of cloudlets is formulated and a balanced mapping of cloudlets is performed. The system architecture of RALBA is depicted in Fig. 2. RALBA comprises of two sub-schedulers i.e., *Fill* and *Spill* schedulers. *Fill* Scheduler performs *Cloudlet* to VM allocation via considering the computing share of VMs. *Fill* scheduler selects $VM_j$ with *Largest_VMShare* and determines $maxPCloudletVM_j$ for $VM_j$. The candidate *Cloudlet* to VM allocation is performed and $VMShare_j$ of $VM_j$ is modified after allocation of the *Cloudlet*. *Fill* scheduler repeats the process of cloudlets mapping until there does not exist $VM_j$ with non-empty $RPCloudlet_j$ or the *CLS* becomes empty.

*Spill* scheduler performs *Cloudlets* to VMs allocation based on EFT of candidate *Cloudlet*. After *Fill* scheduler, RALBA system switches to the *Spill* scheduler to allocate

**Table 2** Preliminary notations used in RALBA system and performance model

| Notations | Descriptions |
|---|---|
| *Cloudlet* | A notation for a job in CloudSim simulator |
| MI | *Million Instructions* (size of a *Cloudlet*) |
| MIPS | *Million Instructions Per Second* (computing power of a VM) |
| *VMS* | Set of VMs in a data-center |
| *vmCrMap* | Set of sorted VMs with its computing ratio |
| $vmCrMap_j$ | Computation Ratio of VM *j* in *vmCrMap* |
| *CLS* | Set of all cloudlets (to be scheduled) |
| $Cloudlet_i$.MI | Size of $Cloudlet_i$ in MI |
| $VM_j.MIPS$ | Computing power of $VM_j$ in MIPS |
| $RPCloudlet_j$ | Set of remaining possible cloudlets that can be assigned to $VM_j$ |
| $maxPCloudletVM_j$ | Largest cloudlet in $RPCloudlet_j$ that is assigned to $VM_j$ |
| *minCloudlet* | Smallest sized *Cloudlet* (to be scheduled) |
| *maxCloudlet* | Largest sized *Cloudlet* (to be scheduled) |
| *VMShare* | Set of sorted VMs with its computing share |
| $VMShare_j$ | Computing share of VM *j* (in MI) in *VMShare* |
| *Largest_VMShare* | Current largest computing share for any VM in *VMS* |
| $Cloudlet\_CT_{ij}$ | Expected completion time of $Cloudlet_i$ on *VM j* |
| $VM\_CT_j$ | Completion time of *VM j* |
| $Cloudlet\_EFT_i$ | Earliest finish time of $Cloudlet_i$ |
| *ARUR* | *Average Resource Utilization Ratio* (ARUR) of *VMS* |
| *Fill_Scheduler* | Sub-scheduler that assigns $Cloudlet_i$ to $VM_j$ based on $VMShare_j$ |
| *Spill_Scheduler* | Sub-scheduler that assigns $Cloudlet_i$ to $VM_j$ based on $Cloudlet\_EFT_i$ |

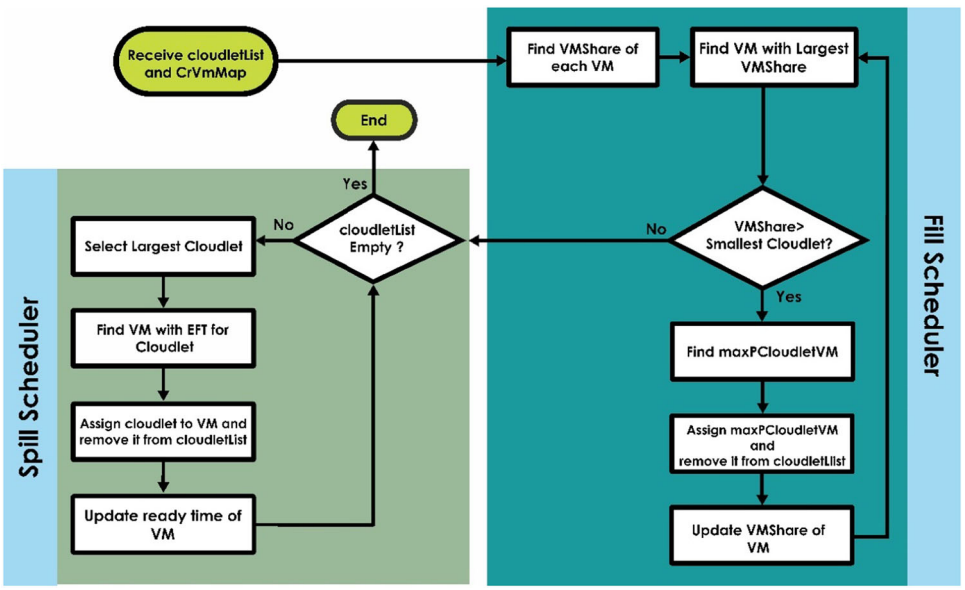



**Fig. 2** RALBA system architecture

the remaining *Cloudlets* of *CLS*. The *maxCloudlet* is selected and allocated to the specific VM that produces EFT for this *maxCloudlet*. On candidate *Cloudlet*–VM allocation, the completion time of the VM is updated. *Cloudlet* to VM allocation is repeated until *CLS* becomes empty.

### 3.3 RALBA system model

A unified Cloud system model is formed to delineate the performance of RALBA based cloudlet scheduling. A Cloud comprises of a set of VMs represented as $VMS = \{VM_1, VM_2, \ldots, VM_m\}$, where m is the total number of VMs and a specific VM can be represented as $VM_j$

($1 \leq j \leq m$). *VMS* represents the computing resources responsible for the execution of cloudlets. The set of cloudlets is presented as $CLS = \{Cloudlet_1, Cloudlet_2, \ldots, Cloudlet_n\}$, where n is the total number of cloudlets and a specified cloudlet can be represented as $Cloudlet_i$ ($1 \leq i \leq n$). A resource manager in Cloud is responsible to provide the computation ratios (*vmCrMap*) of all VMs to RALBA, where $vmCrMap_j$ can be computed as:

$$vmCrMap_j = \left(\frac{VM_j.MIPS}{\sum_{k=1}^{m} VM_k.MIPS}\right) \quad \forall j = \{1, 2, \ldots, m\} \tag{1}$$

*vmCrMap* is used to maintain a balanced workload allocation to VMs and assists in calculating the computing share of all VMs (*VMShare*). The $VMShare_j$ of a specified VM is mathematically expressed as:

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

*Fill* scheduler allocates *Cloudlet* to VM based on *VMShare*. The $RPCloudlet_j$ of a specified $VM_j$ can be computed as:

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

$maxPCloudletVM_j$ in each scheduling decision of *Fill* scheduler can be computed as:

$$maxPCloudletVM_j = \max_{\forall p \in RPCloudletj} Size(p) \tag{4}$$

In addition, on each scheduling decision of *Fill* and *Spill* schedulers, the candidate cloudlet is removed from the list of cloudlets with Eq. (5):

$$CLS = \begin{cases} (CLS) - (maxPCloudletVMj), & Using\ Fill\_Scheduler \\ (CLS) - (maxCloudlet), & Using\ Spill\_Scheduler \end{cases} \tag{5}$$

Equations (6) and (7) computes *minCloudlet* and *maxCloudlet*, respectively.

$$minCloudlet = \max_{\forall c \in CLS} Size(c) \tag{6}$$

$$maxCloudlet = \max_{\forall c \in CLS} Size(c) \tag{7}$$

On cloudlet–VM allocation, $VMShare_j$ of $VM_j$ is modified by *Fill* Scheduler as:

$$VMShare_j = \begin{cases} Switch\ to\ Spill\_Scheduler, VMShare_j < minCloudlet \\ VMShare_j - maxPCloudletVM_j.MI, VMShare_j \geq minCloudlet \end{cases} \tag{8}$$

In each scheduling decision of the *Fill* scheduler, the VM with largest $VMShare_j$ is selected. The *Largest_VMShare* can be computed as:

$$Largest\_VMShare = \max_{\forall j \in \{1,2,3,\ldots,m\}} \left(VMShare_j\right) \tag{9}$$

*Spill* scheduler uses $Cloudlet\_EFT_i$ to allocate the remaining cloudlets to VMs. The $Cloudlet\_EFT_i$ of a specified $Cloudlet_i$ is computed using the mathematical relation:

$$Cloudlet\_EFT_i = \min_{\forall j \in \{1,2,3,\ldots,m\}} \left(Cloudlet\_CT_{ij}\right) \tag{10}$$

where $Cloudlet\_CT_{ij}$ is the expected completion time of $Cloudlet_i$ on $VM_j$. The $Cloudlet\_CT_{ij}$ is mathematically defined and expressed as:

$$Cloudlet\_CT_{ij} = \left(\frac{Cloudlet_i.MI}{VM_j.MIPS}\right) + VM\_CT_j \tag{11}$$

The $VM\_CT_j$ is the completion time of $VM_j$ for already assigned workload prior to the allocation of $Cloudlet_i$ and computed as:

$$VM\_CT_j = \sum_{i=1}^{n} \left(\frac{Cloudlet_i.MI \times F[i,j]}{VM_j.MIPS}\right) \tag{12}$$

where *F[i,j]* is a Boolean variable that determines the allocation of $Cloudlet_i$ to $VM_j$ represented in Eq. (13).

$$F[i,j] = \begin{cases} 1, & cloudlet\ \text{i}\ is\ assigned\ to\ VMj \\ 0, & Otherwise \end{cases} \tag{13}$$

### 3.4 Performance model

This study evaluates RALBA based scheduling using *makespan*, *throughput,* and *resource utilization* performance metrics. The mathematical presentation of makespan is [27, 34]:

$$Makespan = \max_{\forall j \in \{1,2,3,\ldots,m\}} \left(VM\_CT_j\right) \tag{14}$$

Throughput definition is presented in Eq. (15). A higher throughput value indicates high workload execution per unit time.

$$Throughput = \frac{\text{n}}{\text{Makespan}} \tag{15}$$

where, *n* represents total Cloud jobs executed. ARUR is a measure to present the overall utilization of Cloud [27, 34] (expressed in Eq. (16)). The value of ARUR remains between 0 and 1. The higher ARUR value represents higher resource utilization in Cloud.

**Table 3** Variables and their interpretation (used in Algorithms 1—3)

| Variables | Description |
|---|---|
| vmList | List of VMs in a data-center |
| cloudletList | List of cloudlets (to be scheduled) |
| cloudletVmMap | Allocation map (vm, cloudlet) of cloudlets to VMs |
| totalLength | Sum of cloudlet sizes in cloudletList (in MI) |
| vShareMap | Computing share map (vm, share) of VMs |
| $vShareMap_v$ | Computing share of VM $v$ in vShareMap |
| newVShare | Updated computation share of a VM |
| V | An instance of a VM |
| Cloudlet | An instance of a Cloudlet |

$$ARUR = \frac{\left(\frac{\sum_{j=1}^{m} VM\_CT_j}{m}\right)}{Makespan} \tag{16}$$

### 3.5 RALBA algorithm

This section elaborates the proposed scheduling heuristic RALBA (shown in Algorithm 1) with its two primary modules i.e., *Fill* Scheduler (presented in Algorithm 2) and *Spill* Scheduler (shown in Algorithm 3). Table 3 illustrates the interpretations of data items harnessed in Algorithms 1–3.

List of *Cloudlets* and list of all VMs computation ratio (provided by Cloud resource manager) are used as input parameters in RALBA (Algorithm 1). To perform mapping of *Cloudlets* to VMs, RALBA invokes *Fill* scheduler (line 2 of Algorithm 1) and then invokes the *Spill* scheduler (line 5 of Algorithm 1).

Fill Scheduler (Algorithm 2) performs the necessary initializations (lines 1–4) and computes the *totalLength* of all the cloudlets in the submitted batch (lines 5–6). Afterwards, the computing share of each VM ($vShareMap_v$) is calculated (lines 7–8 of Algorithm 2) by using Eq. (2). A while-loop (lines 9–15 of Algorithm 2) is used to select the VM with the largest computation share and to determine the *maxPCloudletVm* for scheduling. On each scheduling decision (lines 13–15 of Algorithm 2), the candidate cloudlet is removed from the cloudletList and the $vShareMap_v$ of the candidate VM is modified (using Eq. (8)). This process (lines 9–15 of Algorithm 2) is repeated until the smallest size cloudlet becomes greater than the largest $vShareMap_v$ in vShareMap.

After cloudlets–VMs allocation by the *Fill* scheduler, the Spill Scheduler (Algorithm 3) is employed for the allocation of the remaining cloudlets to VMs. While-loop (in line 1 of Algorithm 3) allocates the remaining cloudlets (in descending order of cloudlets size) to VMs based on EFT. The *maxCloudlet* is selected on line 2 and the VM producing *Cloudlet_EFT* is determined in line 3. This candidate cloudlet to VM allocation is performed in line 4 within each loop-iteration. At each scheduling decision, the candidate cloudlet is removed from the cloudletList and the $VM\_CT_j$ of $VM_j$ is modified (using Eq. (12)). This process is repeated until *cloudletList* becomes empty. RALBA produces a load-balanced schedule in the form of *cloudletVmMap*.

The source code of the RALBA is available on the *Bitbucket Repository*[1] in a project named *prjRALBA*.

**Algorithm 1:** RALBA

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

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

**Algorithm 2:** FillScheduler

**Input:** $vmCrMap$ - set of VMs with computation ratio ,
$cloudletList$ - list of cloudlets $c_1, c_2, \ldots, c_n$
**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* $cloudletList$ **do**
6 $\quad totalLenght$ = totalLenght + cloudlet.getCloudletLength()
7 **forall** $v$ *in* $vmList$ **do**
8 $\quad vShareMap_v$ = totalLenght * $vmCrMap_v$
9 **while** $getMinCloudlet(cloudletList) \geq getLargeShare(vShareMap)$ **do**
10 $\quad v = getLargeShareVm$(vShareMap)
11 $\quad cloudlet = getMaxPCloudletVm$(v,cloudletList)
12 $\quad cloudletVmMap.add$(cloudlet,v)
13 $\quad newVShare = vShareMap.get$(v)− cloudlet.$getCloudletLength$
14 $\quad vShareMap.modify$(v,newVShare)
15 $\quad cloudletList.remove$(cloudlet)
16 **return** $cloudletVmMap$

**Algorithm 3:** SpillScheduler

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

1 **while** $cloudletList.size() \geq 1$ **do**
2 $\quad cloudlet = getMaxCloudlet$(cloudletList)
3 $\quad v = getVmWithEFT$(cloudlet,vmList)
4 $\quad cloudletVmMap.add$(cloudlet,v)
5 $\quad cloudletList.remove$(cloudlet)
6 **return** $cloudletVmMap$

[1] https://bitbucket.org/RALBA_18/ralba/src.

**Table 4** Computational complexity of scheduling heuristics

| Heuristics | RR and RS [7] | MCT [7] | Min–Min, Max–Min, RASA, TASA, and Sufferage [18, 29] | RALBA |
|---|---|---|---|---|
| Complexity | $O(N)$ | $O(MN)$ | $O(MN^2)$ | $O(M^2n + M.N–n)$ |

**Table 5** Overhead analysis of scheduling heuristics

| Heuristics | RS | MCT | RALBA | Max–Min | RASA | Min–Min | TASA | Sufferage |
|---|---|---|---|---|---|---|---|---|
| Scheduling overhead (*N times of RR*) | 1.01 | 1.24 | 1.43 | 2.05 | 3.45 | 6.40 | 11.11 | 13.36 |

**Table 6** Configuration of the simulation environment

| | |
|---|---|
| Simulator/version | CloudSim version 3.0.2 |
| Computing power of cloud host machines | 04 Dual-core (4000 MIPS), 26 Quad-core (4000 MIPS) |
| Total cloud host machines | 30 |
| Host machine memory | 16,384 MBs each |
| Total VMs | 50 heterogeneous VMs (as shown in Fig. 3) |
| Total cloudlets | 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000 |

### 3.6 Complexity and overhead analysis

This section delineates the complexity and overhead analysis of the scheduling heuristics. To scrutinize the complexity of RALBA, we consider $N$ as the total number of cloudlets and $M$ as the total number of VMs in a Cloud data-center. In worst-case, *Fill* scheduler selects a VM with largest *VMShare* using $M$ number of comparisons and determines *maxPCloudletVM* after $N$ comparisons. After cloudlet–VM allocation, the $vShareMap_v$ is modified in *vShareMap* with a maximum of $M$ comparisons. The time complexity of *Fill* scheduler is $O(M^2N)$; where $M \ll N$ on the real Cloud. On the other hand, *Spill* scheduler selects the *maxCloudlet* from a sorted *cloudletList* and assigns to the VM producing EFT for it. In worst-case, when $N$ cloudlets are scheduled by *Spill* scheduler, the time complexity of *Spill* scheduler is $O(MN)$. If $n$ is the number of cloudlets scheduled by *Fill* scheduler, then remaining $N–n$ cloudlets will be scheduled by *Spill* scheduler. Therefore, the computational complexity of RALBA becomes $O(M^2.n + M.N–n)$. Table 4 provides a comparison of the computational complexities of RALBA and the existing scheduling heuristics.

Furthermore, we profile the simulation code of RALBA and other existing scheduling heuristics to collect the time overhead related to the scheduling decisions. RR presents the minimal scheduling overhead against the other heuristics. Table 5 delineates the relative scheduling overhead in terms of complexity order ($N$ times of RR scheduling) from best to worst i.e., *RS*, *MCT*, *RALBA*, *Max–Min*, *RASA*, *Min–Min*, *TASA*, and *Sufferage*. The Sufferage heuristic has the most expensive time overhead among the given set of heuristics. Considering the computational complexity (Table 4) and the overhead analysis (Table 5), RALBA is promised to schedule jobs in a scalable manner for large-dataset.

## 4 Experimental evaluation and discussions

This section encompasses the experimental evaluation of RALBA as compared to the other scheduling heuristics.

### 4.1 Experimental setup

For empirical evaluation, we employ a renowned Cloud simulator *CloudSim* [35]. It is an open-source framework for modeling and performance analysis of Cloud environment and services. The experiments are performed on a machine equipped with Intel Core i3-4030U Quad-core processor (having 1.9 GHz clock speed) and 04 GBs of main memory. Table 6 illustrates the configuration detail for the employed simulation environment. All the experiments are performed by using 50 VMs, hosted on 30 host machines within a data-center. Figure 3 presents the overall statistics of the VMs and their computing power in

terms of *Millions of Instructions Per Second* (MIPS). As shown in Fig. 3, the slowest and fastest VMs have the computing power of 100 and 4000 MIPS, respectively.

### 4.2 Workload generation

For experiments, two workloads of independents cloudlets are generated using the guidelines available in the literature [35–38], i.e., (i) *synthetic* workload (ii) *Google-like* realistic workload. The synthetic workload is created using random-number generation mechanism employing 05 different cloudlet-size ranges i.e., *tiny* (1–250 MI), *small* (800–1200 MI), *medium* (1800–2500 MI), *large* (7000–10,000 MI), and *extra-large* (30,000–45,000 MI) (as shown in Fig. 4).

Using *Monte-Carlo* simulation method [39], the Google-like workload is generated based on the realistic Google cluster traces. For the creation of real-world traces, analysis of Google cluster traces [38, 40–46] and MapReduce logs from the M45 supercomputing cluster [44] was performed. The analysis affirms that the majority of cloudlets were of small (execution time less than 15 min) and few large size cloudlets (exceeding the execution time over 300 min) [38]. Based on the analysis, we formulate the following cloudlet sizes for the Google-like realistic workload: *small* (15,000–55,000 MI), *medium* (59,000–99,000 MI), *large* (101,000–135,000 MI), *extra-large* (150,000–337,500 MI), and *huge* (525,000–900,000 MI) (as shown in Fig. 5).

### 4.3 Simulation results

The performance of RALBA and the other Cloud scheduling heuristics (i.e., *TASA*, *Sufferage*, *Max–Min*, *RASA*, *Min–Min*, *MCT*, *RR*, and *RS*) is compared by using

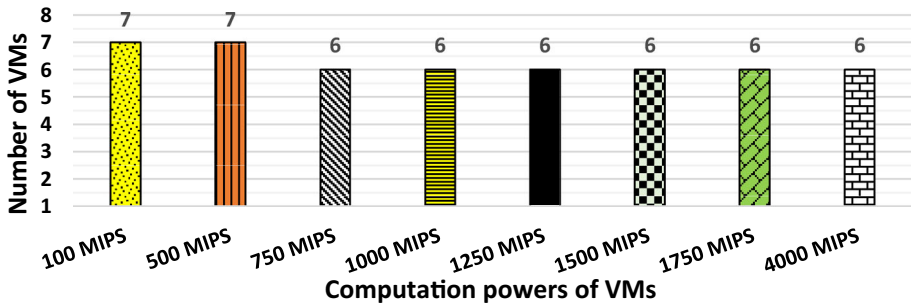



**Fig. 3** Computation power of heterogeneous VMs

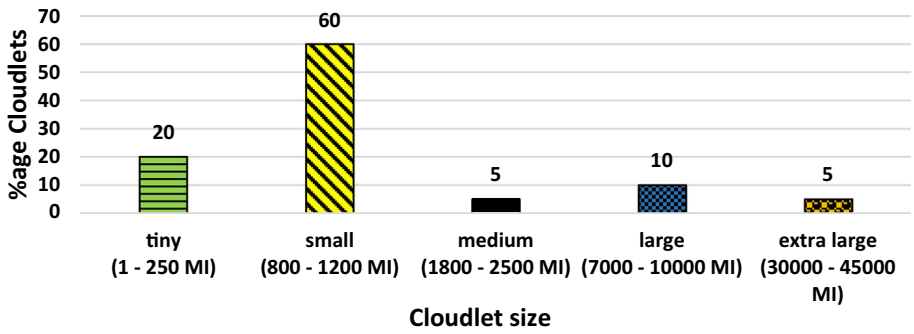



**Fig. 4** Composition of *synthetic* workload

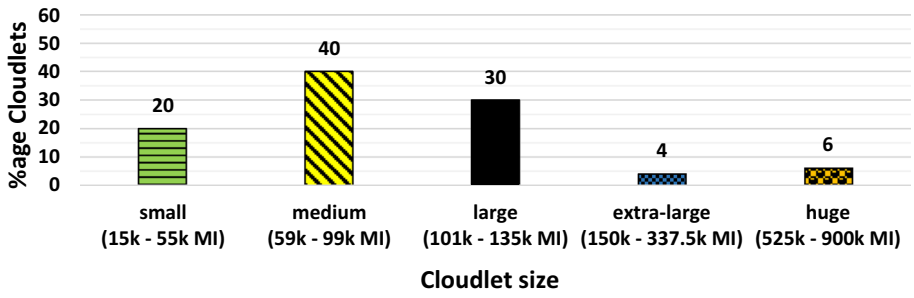



**Fig. 5** Composition of *Google-like* realistic workload

makespan, *Average Resource Utilization Ratio* (ARUR), and throughput metrics. Each experiment is performed 05 times and the analysis is conducted on average values.

Figure 6 presents the average makespan based results for the synthetic workload. Figure 6 shows that RALBA consumes on average 5.5, 6.9, 243.9, 246.4, 46.1, and 41.9% less time for the execution of synthetic workload as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, and MCT heuristics, respectively. For the execution of Google like realistic workload (shown in Fig. 7), RALBA consumes on average 3.8, 1.8, 47.5, 24.5, 28.1, and 18.6% lower time as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, and MCT heuristics, respectively.

Figure 8 presents the mean ARUR based experimental results for execution of the synthetic workload. Figure 8 shows that RALBA has attained 10.9, 9.8, 103.3, 117.6, 121.6, 65, 624.1, and 541.2% higher resource utilization as compared to TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR heuristics, respectively. For the execution of Google like realistic workload, RALBA has achieved higher resource utilization of 16.9, 7.4, 15.9, 13.9, 78.8, 30.1, 550.6, and 541.7% as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR scheduling heuristics, respectively (see Fig. 9).

Figure 10 presents average throughput results for execution of the synthetic workload. As shown in Fig. 10, RALBA has achieved 6.2, 7.5, 151.6, 160.1, 47.2, 42.1, 2047, and 1812% higher throughput as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR scheduling heuristics, respectively. Figure 11 shows the average throughput results for Google like realistic workload. As shown in Fig. 11, RALBA has

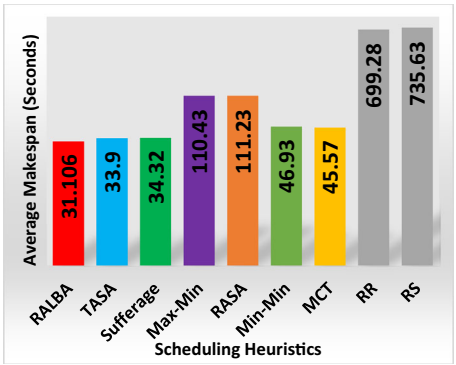



**Fig. 6** Average makespan—synthetic workload

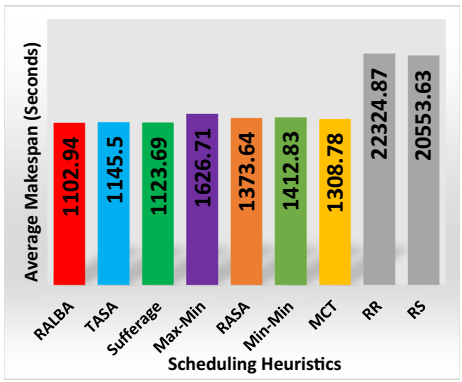



**Fig. 7** Average makespan—Google-like workload

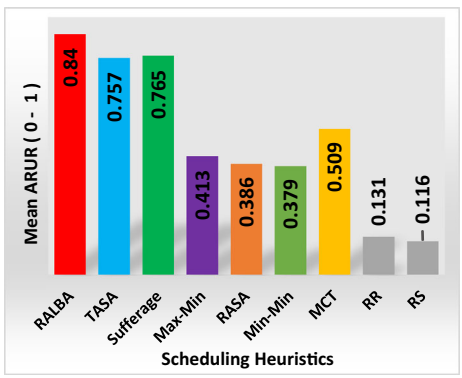



**Fig. 8** Mean ARUR—synthetic workload

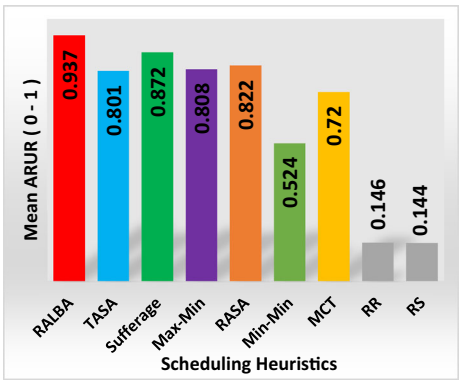



**Fig. 9** Mean ARUR—Google-like workload

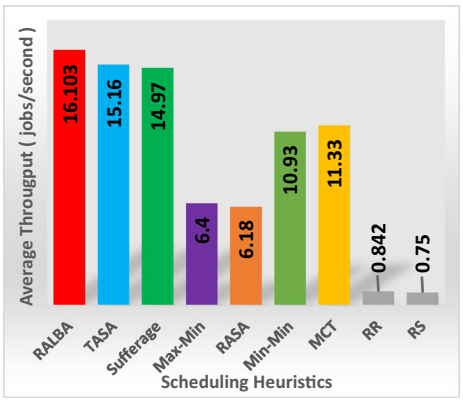



**Fig. 10** Average throughput—synthetic workload

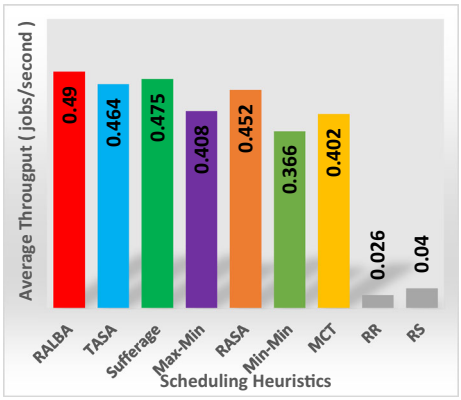



**Fig. 11** Average throughput—Google-like workload

attained higher job execution throughput of 5.6, 3.1, 20.1, 8.4, 33.8, 21.9, 1125, and 1785% as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR heuristics, respectively.

### 4.4 Results and discussion

It can be scrutinized that the proposed scheduling heuristic has successively achieved significant improvements in terms of makespan and resource utilization for the *skewed* workload [16] (especially the *positively skewed* one) because of the inherent resource aware scheduling mechanism employed by RALBA. We refer workload having a large number of shorter cloudlets (*positively skewed*) and a large number of longer size cloudlets (*negatively skewed*). A modest improvement in makespan and significant higher resource utilization is observed for execution of non-skewed workload. The higher resource utilization achieved by RALBA is due to the resource aware mapping that results in a load balanced execution of the workload.

The composition of the synthetic workload (as shown in Fig. 4) shows that there are a large number of smaller and a few outsized (i.e., 10% large and 05% extra-large) cloudlets. This workload composition shows that the synthetic workload is a more positively skewed as compared to the Google-like realistic workload (i.e., 05% more number of large cloudlets). Experiments have revealed that the MCT scheduling heuristic reduces the makespan; however, it causes an imbalanced mapping of cloudlets by overloading the faster VMs. The execution of the synthetic workload has resulted in more imbalanced mapping as compared to the scheduling of Google like realistic workload which arose due to the more positively skewed nature of the synthetic workload as compared to the Google-like workload.

The experimental evaluation has revealed that the Min–Min has acquired reduced makespan as compared to the Max–Min due to a large number of shorter size cloudlets in both workloads (i.e., synthetic and Google like). The inherent scheduling mechanism of Max–Min maps some larger cloudlets to slow VMs producing a longer makespan [21]. As compared to Min–Min heuristic, RALBA has achieved 28.1% (for Google like realistic workload) and 46.1% (for synthetic workload) reduced makespan. Moreover, RALBA has consumed 243.9 and 47.5% reduced time as compared to Max–Min for execution of the Google-like and synthetic workloads, respectively. Max–Min has produced longer makespan (for Google-like realistic workload) due to few huge size cloudlets as compared to the synthetic workload. Alternatively, Max–Min has significantly improved the resource utilization as compared to the Min–Min that overloads faster VMs (producing load imbalance). However, Max–Min has not significantly improved the resource utilization (for execution of the synthetic workload) as compared to the Google-like workload because the synthetic workload contains high number of large size cloudlets.

Our experimental evaluation exhibits that the Sufferage scheduling heuristic produces lower makespan and higher resource utilization as compared to the Min–Min, Max–Min, and RASA heuristics. This reduced makespan and higher resource utilization are achieved due to the selection of a *suitable* VM for cloudlet mapping. The suitable VM represents the computing resource which will suffer most (in terms of makespan) if the candidate cloudlet has not been mapped to that resource (in the current scheduling iteration).

The VM–cloudlet allocation mechanism of TASA is based on Sufferage and Min–Min heuristics. Therefore, TASA provides reduced makespan among the existing heuristics. Figure 7 shows that RALBA has produced slightly reduced makespan as compared to TASA and Sufferage for the Google-like workload (i.e., 1.88 and 3.86% reduced, respectively). TASA and Sufferage have performed almost identical to RALBA due to the inherent resource-aware mapping mechanism employed by both of these scheduling heuristics. However, RALBA also considers load-balance factor for scheduling; therefore, it has produced higher resource utilization (i.e., 7.45 and 16.97% more resource utilization as compared to TASA and Sufferage, respectively) for Google like workload as well. It is evident that the TASA has produced lower resource utilization (as compared to the Sufferage) due to the inherent use of Min–Min scheduling mechanism.

The scheduling mechanism of RASA is based on Max–Min and Min–Min heuristics. Therefore, RASA provides improved resource utilization over TASA for the Google-like workload (better resource utilization due to the inherent Max–Min mechanism). Overall scrutinization of the conducted experiments persuades that RALBA has achieved significant improvement in resource utilization,

makespan, and throughput as compared to the existing scheduling heuristics. However, RALBA does not support SLA-aware scheduling of Cloud jobs. Therefore, the SLA based resource (e.g., execution-cost, bandwidth, memory, etc.) and deadline constrained cloudlets may not be scheduled adequately.

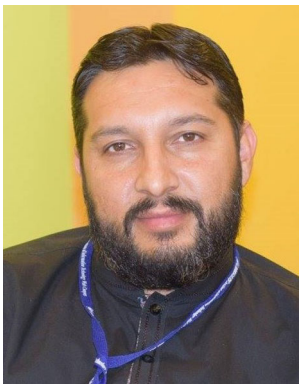
## 5 Conclusions and future work

In-depth analysis of the current state-of-the-art scheduling heuristics results in inefficient resource utilization, reduced makespan, and low throughput due to the imbalanced mapping of Cloud jobs. This study presents a novel Cloud scheduling heuristic RALBA that ensures improved resource utilization with minimal makespan and increased throughput. The performance of RALBA has been compared with the state-of-the-art scheduling heuristics in terms of makespan, resource utilization, and throughput. The detailed analysis of experimental results has shown that RALBA has successively attained lower makespan, higher throughput, and improved resource utilization as compared to the existing Cloud scheduling heuristics (i.e., TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR). In future, we intend to enhance the functionalities of RALBA to deal with the unexpectedly slow computing resources and to cope with the sudden resource failures with a fault-tolerant scheduling mechanism.

## References


1. Ray, P.P.: The Green grid SAGA—A Green initiative to data centers: a review. Indian J. Comput. Sci. Eng. **1**(4), 333–339 (2012)
2. Hayes, B.: Cloud computing. Commun. ACM **51**(7), 9 (2008)
3. Rimal, P., Choi, E., Lumb, I.: A taxonomy and survey of cloud computing systems. In: NCM 2009—5th International Joint Conference INC, IMS, IDC, pp. 44–51 (2009)
4. Fernandez-baca, D.: Allocating modules to processors in a distributed system. IEEE Trans. Softw. Eng. **15**(11), 1427–1436 (1989)
5. Cook, S.A.: The complexity of theorem-proving procedures. In: in: Proceedings of 3rd Annual ACM Symposium on Theory of Computing, pp. 2–8 (1971)
6. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof system. SIAM J. Comput. **18**(1), 186–208 (1989)
7. Braun, T.D., et al.: 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 (2001)
8. Ibarra, O.H., Kim, C.E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. J. ACM **24**(2), 280–289 (1977)
9. Deldari, A., Naghibzadeh, M., Abrishami, S.: CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud. J. Supercomput. **73**(2), 756–781 (2016)
10. Kumar, S., Nadjaran, A., Gopalaiyengar, S.K.: SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter. J. Netw. Comput. Appl. **45**, 108–120 (2014)
11. Jennings, B., Stadler, R.: Resource management in clouds: survey and research challenges. J. Netw. Syst. Manag. **23**(3), 567–619 (2014)
12. Wu, F., Wu, Q., Tan, Y.: Workflow scheduling in cloud: a survey. J. Supercomput. **71**(9), 3373–3418 (2015)
13. Farrag, A.A.S., Abbas, S., El-Horbaty, E.-S.M.: Intelligent cloud algorithms for load balancing problems: a survey. In: IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS), December 2015, pp. 210–216 (2015)
14. Lenzini, L., Mingozzi, E., Stea, G.: Tradeoffs between low complexity, low latency, and fairness with deficit round-robin schedulers. IEEE/ACM Trans. Netw. **12**(4), 681–693 (2004)
15. Chen, H., Wang, F., Helian, N., Akanmu, G.: 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 (2013)
16. Panda, S.K., Agrawal, P., Khilar, P.M., Mohapatra, D.P.: Skewness-based min–min max–min heuristic for grid task scheduling. In: Proceedings of the 2014 Fourth International Conference on Advanced Computing & Communication Technologies, pp. 282–289 (2014)
17. Hung, T.C., Phi, N.X.: Study the effect of parameters to load balancing in cloud computing. Int. J. Comput. Netw. Commun. **8**(3), 33–45 (2016)
18. Yu, X., Yu, X.: A new grid computation-based min–min algorithm. In: Sixth International Conference on Fuzzy Systems and Knowledge Discovery, pp. 43–45 (2009)
19. Mao, Y., Chen, X., Li, X.: Max–min task scheduling algorithm for load balance in cloud computing. In: Proceedings of International Conference on Computer Science and Information Technology, vol. 255, pp. 457–465 (2014)
20. Muhammed, A., Abdullah, A., Hussin, M.: Max-average: an extended max–min scheduling algorithm for grid computing environment. J. Telecommun. Electron. Comput. Eng. **8**(6), 43–47 (1843)
21. Tabak, E., Cambazoglu, B., Aykanat, C.: Improving the performance of independent task assignment heuristics minmin, maxmin and sufferage. IEEE Trans. Parallel Distrib. Syst. **25**(5), 1244–1256 (2014)
22. Aditya, A., Chatterjee, U., Gupta, S.: 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 (2015)
23. Mohialdeen, I.A.: Comparative study of scheduling algorithms in cloud computing environment. J. Comput. Sci. **9**(2), 252–263 (2013)
24. Tchernykh, A., et al.: Online Bi-Objective Scheduling for IaaS Clouds Ensuring Quality of Service. J. Grid Comput. **14**(1), 5–22 (2016)
25. Elzeki, O.M., Rashad, M.Z., Elsoud, M.A.: Overview of scheduling tasks in distributed computing systems. Int. J. Soft Comput. Eng. **2**(3), 470–475 (2012)
26. Li, B., Pei, Y., Wu, H., Shen, B.: Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds. J. Supercomput. **71**(8), 3009–3036 (2015)
27. Biradar, S., Pawar, D.: A review paper of improving task division assignment using heuristics. Int. J. Sci. Res. **4**(1), 609–613 (2015)
28. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic mapping of a class of independent tasks onto

heterogeneous computing systems. J. Parallel Distrib. Comput. **59**(2), 107–131 (1999)
29. Parsa, S., Entezari-Maleki, R.: RASA: a new grid task scheduling algorithm. Int. J. Digit. Content Technol. Appl. **3**(4), 152–160 (2009)
30. Sharma, G., Banga, P.: Task aware switcher scheduling for batch mode mapping in computational grid environment. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **3**, 1292–1299 (2013)
31. Mathew, T.: Study and analysis of various task scheduling algorithms in the cloud computing environment. In: IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 658–664 (2014)
32. Patel Dhara, R., Thaker, C.: Analysis of various task scheduling algorithms in cloud computing. Int. J. Sci. Res. Sci. Eng. Technol. **1**(6), 245–249 (2015)
33. Dehkordi, S.T., Bardsiri, V.K.: TASA: a new task scheduling algorithm in cloud computing. J. Adv. Comput. Eng. Technol. **1**(4), 25–32 (2015)
34. Panda, S.K., Jana, P.K.: SLA-based task scheduling algorithms for heterogeneous multi-cloud environment. J. Supercomput. **73**(6), 2730–2762 (2017)
35. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **39**(7), 701–736 (2009)
36. Mehdi, N.A., Mamat, A., Ibrahim, H., Subramaniam, S.K.: Impatient task mapping in elastic cloud using genetic algorithm. J. Comput. Sci. **7**(6), 877–883 (2011)
37. Behzad, S., Fotohi, R., Effatparvar, M.: Queue based job scheduling algorithm for cloud computing. Int. J. Basic Appl. Sci. **4**(12), 3785–3790 (2013)
38. Liu, Z., Cho, S.: Characterizing machines and workloads on a Google cluster. In: 41st International Conference on Parallel Processing Workshops, pp. 397–403 (2012)
39. Chen, Y., Katz, R.H.: Analysis and Lessons from a Publicly Available Google Cluster Trace. EECS Dep. Univ. California, Berkeley, Tech. Rep. UCB/EECS-2010-95 94, p. 11 (2010)
40. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Towards understanding heterogeneous clouds at scale: Google trace analysis. Intel Sci. Technol. Cent. Cloud Comput. Tech. Rep. ISTC-CC-TR-12-101
41. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale. In: Proceedings of the Third ACM Symposium on Cloud Computing—SoCC'12, pp. 1–13 (2012)
42. Moreno, I.S., Garraghan, P., Townend, P., Xu, J.: An approach for characterizing workloads in google cloud to derive realistic resource utilization models. In: Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, pp. 49–60 (2013)
43. Liu, C., Liu, C., Shang, Y., Chen, S., Cheng, B., Chen, J.: An adaptive prediction approach based on workload pattern discrimination in the cloud. J. Netw. Comput. Appl. **80**, 35–44 (2017)
44. Kavulya, S., Tany, J., Gandhi, R., Narasimhan, P.: An analysis of traces from a production MapReduce cluster. In: 11th IEEE/ACM International Conference on Grid Computing (CCGrid), pp. 94–103 (2010)
45. Elzeki, O.M., Reshad, M.Z., Elsoud, M.A.: Improved max–min algorithm in cloud computing. Int. J. Comput. Appl. **50**(12), 22–27 (2012)
46. De Falco, I., Scafuri, U., Tarantino, E.: Two new fast heuristics for mapping parallel applications on cloud computing. Futur. Gener. Comput. Syst. **37**, 1–13 (2014)



**Altaf Hussain** received the M.S. degree in computer software engineering from National University of Science and Technology (NUST), Islamabad, Pakistan. He received his B.S. in computer science with distinction from NWFP AUP, Pakistan. His research interests include software testing, data mining and distributing computing comprises performance analysis and Cloud computing. He is currently a Ph.D. scholar at Capital University of Science and Technology, Islamabad, Pakistan.

**Muhammad Aleem** received the Ph.D. degree in computer science from the Leopold-Franzens-University, Innsbruck, Austria in 2012. His research interests include parallel and distributed computing comprise programming environments, multi-/many-core computing, performance analysis, cloud computing, and big-data processing. He is currently working as assistant professor at Capital University of Science and Technology, Islamabad, Pakistan.

**Abid Khan** is working as an assistant professor in computer science department at COMSATS Institute of Information Technology (CIIT), Islamabad. He was a postdoc fellow at Politecnico de Torino, Italy from 2009 to 2011. He did his Ph.D. from Harbin Institute of technology, Harbin, P.R. China in 2008. His research interest includes applied cryptography, security and privacy issues in distributed systems, secure provenance. He has more than 12 years of teaching, research and development experience.

**Muhammad Azhar Iqbal** is an assistant professor at the Capital University of Science and Technology, Islamabad, Pakistan. He received Ph.D. degree in communication and information systems from the Huazhong University of Science and Technology, Wuhan, P.R. China in 2012. His research interests include: coding-aware routing in vehicular ad hoc networks, energy-efficient MAC for wireless body area networks, large-scale simulation modeling and analysis of computer networks in Cloud.

**Muhammad Arshad Islam** completed his doctorate from University of Konstanz, Germany in 2011. His dissertation is related to routing issues in opportunistic network. His current research interests are related to MANETs, DTNs, social-aware routing and graph algorithms. He is currently working as an assistant professor at Capital University of Science and Technology, Islamabad, Pakistan.