Noname manuscript No.
(will be inserted by the editor)

RADL: A Resource and Deadline-aware Dynamic Load-balancer for Cloud Tasks

Said Nabi · Muhammad Aleem* · Masroor Ahmed · Muhammad Arshad Islam · Muhammad Azhar Iqbal

Abstract Cloud service providers acquire the computing resources and allocate them to their clients. To effectively utilize the resources and achieve higher user satisfaction, efficient task scheduling algorithms play a very pivotal role.A number of task scheduling technique have been proposed in the literature. However, majority of these scheduling algorithms fail to achieve efficient resource utilization that causes them to miss tasks deadlines. This is because these algorithms are not resource and deadline-aware. In this research, a Resource and deadline Aware Dynamic Load-balancer (RADL) for Cloud tasks has been presented. The proposed scheduling scheme evenly distribute the incoming workload of compute-intensive and independent tasks at run-time. In addition, RADL approach has the capability to accommodate the newly arrived tasks (with shorter deadlines) efficiently and reduce task rejection. The proposed scheduler monitors/updates the task and VM status at run-time. Experimental results shows that the proposed technique has attained up to 67.74%, 303.57%, 259.2%, 146.13%, 405.06%, and 259.14% improvement for average resource utilization, meeting tasks deadlines, lower makespan, task response time, penalty cost, and task execution cost respec-

Said Nabi
Capital University of Science & Technology,
Islamabad 44000, Pakistan

*Muhammad Aleem
National University of Computer & Emerging Sciences,
Islamabad 44000 Pakistan
E-mail: m.aleem@nu.edu.pk

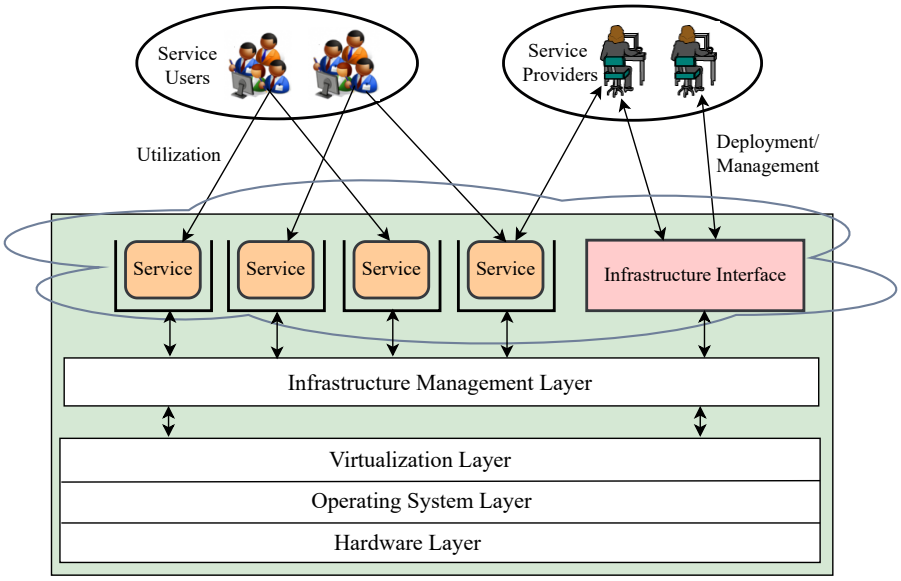


Fig. 1: Cloud Actors [4]

tively as compared to the state-of-the-art tasks scheduling heuristics using three benchmark datasets.

Keywords Cloud · Task scheduling · Dynamic · Resource utilization · Deadline · heuristic · Resource-aware · Cost

1 Introduction

The fast development of storage technologies and processing owing to the Internet has made the computing resources powerful, easily available, and economical to use [1]. This rapid growth in technology has resulted in the birth of Cloud computing [1] that enables end users to share resources like CPU and storage for a particular time based on their needs. Cloud computing model comprises of two key actors [3] (as shown in Figure 1). 1) *Cloud Service Provider*(CSP): deploy the resources like storage, processors, network etc., 2) Service users/clients: hire the services provided by the service providers for their temporal needs. Service providers use internet based-interfaces to make their services accessible to the service users.

In the last few years, Cloud computing has gained high impact on the industry of Information Technology [1]. According to *Gartner Predictions* [7], the market revenue of Cloud infrastructure services will grow by 176% in 2021. Due to such benefits of Cloud [8], many companies like Microsoft [9], Google [10], and Amazon [11] have started providing powerful, reliable, and cost-efficient services [12,13] to the customers. These services include Infrastruc-

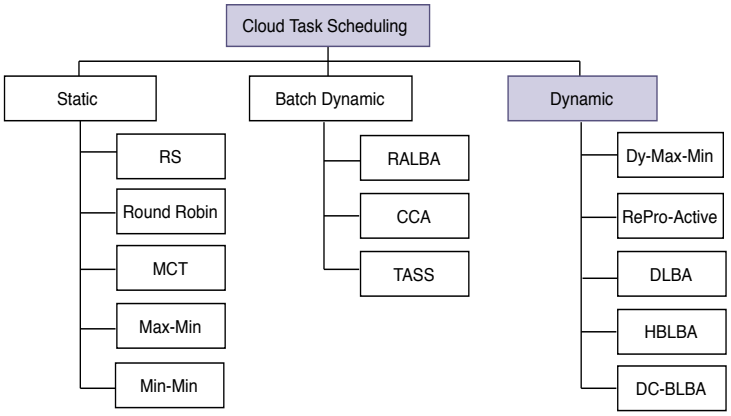


Fig. 2: Types of Cloud Task Schedulers

ture, Platform, and Software which are known as *Software-as-Services* (SaaS), *Platform-as-Services* (PaaS), and *Infrastructure-as-Services* (IaaS) [3,15]. For better delivery of cloud services, efficient utilization of cloud resources is essential [17]. However, the selection of a suitable scheduling algorithm for achieving higher resource utilization is an important and challenging task. Task scheduling heuristics map a job or task [16] on appropriate resource. However, to map large number of tasks on limited resources, task scheduling becomes an NP-hard problem [17–19,49].

Additionally, cloud computing framework provide resizing of the virtualized hardware resources which need dynamic reconfiguration in an automatic manner using infrastructure management interface [4,6]. Cloud mapping can be divided as mapping of VMs on physical host machines and mapping of task on VMs. This research focus on tasks-VMs mapping, where N number of tasks are assigned to VMs with pre-determined computation capability. To map tasks to VMs, there is possibility that a task with high computation requirements can be assigned to the slower VM that can lead to higher execution time which may results in deadline violation [20]. Moreover, a faster VM can get tasks with smaller size may cause the large size tasks to wait for the execution of smaller task which may lead deadline violation for larger tasks. These challenges may imbalance the VMs load and can degrade the overall cloud performance [21, 48]. Load balanced [18,21] cloud task scheduling [49,50] plays a key role to enable efficient use of the Cloud resources. Cloud task scheduling approaches (as shown in Figure 2) are categorized into three types.

1. *Static scheduling* [39,47] is the simplest type of scheduling that maps all the tasks before starting their execution. All information regarding the computing resources and jobs like VMs MIPS and Task size in MI are available in advance for tasks to resource allocation.
2. *Batch dynamic scheduling* techniques [5,21,26] assign a batch of incoming tasks to predefined number of VMs. All tasks in a single batch are statically

allocated to the virtual machines before their execution starts. The number of VMs can be changed (decreased or increased) for the next batch based on the computation requirements of a batch. These task mapping approaches result in issues like delayed response time for the new batch formation (i.e., the task which arrives first will have to wait till the formation of the complete batch) and inter-batch under-utilization of resources (i.e., in circumstances where the execution of the first batch is finished and the next batch not formed yet).

3. *Dynamic schedulers* have the capability to check, estimate, and update the VMs load during the execution of the tasks either in the reactive or proactive manner [17]. These schedulers can allow prioritization and migration [23,27] of the already assigned tasks. Moreover, the dynamic scheduling mechanisms generally have the capability of new VMs creation [18,24,46], removal the existing VMs and migration of VMs [28] at run time.

1.1 Gap Analysis

In static scheduling, once a set of tasks are mapped to the corresponding VMs, their mapping will remain unaltered until the completion of the assigned tasks. Moreover, most of the classical [1,2,20,37] and state-of-the-art heuristic based static task scheduling schemes [5,17,21,23–25] suffer from issues like load imbalance, poor resource utilization, and unable to meet the deadline constraints for the real-time tasks.

In batch dynamic scheduling, tasks of new batches are allocated to VMs without considering the current workload of VMs. Therefor, the newly created VMs for the latest batch can finish the assigned workload earlier than the existing VMs that are busy in executing the previous load. In batch dynamic based approaches, if a task with shorter deadline arrives in later batches; and all the VMs are already overloaded, the newly arrived task need to wait until the completion of previously mapped tasks. Majority of the batch dynamic scheduling heuristics [17,24–27] provide dynamism at the batch level only and suffer from issues related to batch formation delays, and under-utilization of resources (during the batch formation).

Dynamic scheduling mechanisms are much more flexible than the previously described categories [20,39]. However, a number of the existing dynamic scheduling approaches [20] still use an interval-based temporal batch of input tasks that arrived during a particular time period. Majority of the existing dynamic algorithms [17,18,23] suffer from the issues like lower resource utilization, load imbalance, and high rejection ratio for the deadline-based tasks [17,19,23,29]. Moreover, most of these approaches do not consider tasks deadlines [23,24] and lack the task shuffling mechanism to accommodate deadlines.

These issues lead to cloud users dissatisfaction due to high task rejection, maximized task response time, and high tasks execution time (makspan). Moreover, this results in lower Return on Investment (ROI) for cloud service provider due poor utilization of cloud resources. These issues also increase task execution

cost and energy consumption.

Research Questions

1. How to improve cloud resource utilization and load balancing with reduced makespan using heuristic based dynamic scheduler?
2. How to minimize the response time of newly arrived cloud tasks?
3. How to reduce task penalty cost by reducing task rejection and task execution cost by early execution of user tasks?

In this work, we argue that task scheduling algorithms should be resource-aware to achieve better performance. Resource-aware [21,22] load balancing process includes resource discovery, monitoring the current loads on each resource, and assessing workload to be assigned to a resource. To overcome the load imbalance issue, the tasks should be scheduled on the most suitable VMs considering the resource capacities. Therefore, this research proposes a dynamic resource-aware scheduling algorithm named, *Resource and deadline Aware Dynamic Load-balancer* for Cloud tasks (RADL) to mitigate the load imbalance issue and to support deadline constraints. RADL scheduling heuristic distributes the incoming workload of independent, non-preemptive, and compute-intensive tasks in a balanced way. The RADL approach dynamically updates two metrics, i.e., VM-load and ready-time.
The key role of the proposed RADL scheme is to increase resource utilization, meet deadlines of newly arrived deadlines-based tasks, reduce makespan, penalty, and task execution cost [30]. The proposed approach consists of two schedulers: 1) RADL Scheduler: assigns incoming tasks to the pre-defined number of VMs based on the minimum completion time of the tasks. Moreover, the *RADL Scheduler* monitors, updates, the task and VM status at run-time. 2) *Locator Scheduler* (LR-subScheduler), this sub-scheduler places the incoming tasks with a shorter deadline in a suitable position of the task queue of a VM (which provides minimum completion time for the execution of that job).
This mechanism helps to execute the newly arrived tasks (with shorter deadlines) in a timely manner. The proposed scheduling mechanism accommodates the newly arrived deadline-based tasks without migrating (the executing jobs) with no additional VM requirement. The employed mechanism results in the reduced scheduling overhead, high user satisfaction, better Return on Investment (RoI), lower penalty, and total cost. Major contributions of RADL scheduler are summarized as:

- In-depth critical analysis of static and dynamic state-of-the-art scheduling heuristics to investigate their strengths and limitations.
- A novel, dynamic, and load balancing task scheduler for independent, compute-intensive, and non-preemptive Cloud tasks that produce improved resource utilization, reduced task rejection, lower makespan, and higher response time.
- The proposed approach also reduce task penalty cost by reducing task rejection and task execution cost by early execution of user tasks.

– The RADL scheduler is empirically investigated and their performance is evaluated against their counterparts.

Rest of the document is organized as follows. Section 2 discuss the literature review and Section 3 describe system model and architecture, RADL algorithm, complexity analysis, and scheduling overhead. The experimentation, performance evaluation, and discussion are described in Section 4. The paper is concluded in Section 5. Section 5 also discusses the potential future directions.

2 Related Work

In this Section, an extensive literature review of state-of-the-art Cloud tasks scheduling algorithms belonging to the three categories discussed above has been presented.

2.1 Static scheduling heuristics

Static task scheduling heuristics map all the tasks to VMs before starting tasks execution and the tasks once mapped to VMs cannot be altered during the tasks execution. The newly arrived tasks will have to wait for scheduling until the execution of already mapped tasks is completed. One of the simplest heuristic is *Random Selection* (RS) that allocates tasks to VMs in an arbitrary manner without considering the VMs computation power and already assigned workload [21,33]. RS has a simple implementation, low complexity, and minimal scheduling overhead as compared to other scheduling heuristics. RS scheduling algorithm randomly selects VMs and assigns tasks to the selected VMs which overloads some of the VMs leading to load imbalance, poor resource utilization, and longer waiting time [21] for other jobs.
Another basic heuristic is known as *Round Robin* (RR) [34] that distributes the incoming workload in circular order on predefined number of VMs. RR has simple implementation, lower complexity, and minimum scheduling overhead than the other task scheduling algorithms. However, RR assigns tasks to the VMs in circular order irrespective of the VM computation power and task size (causing load imbalance) [21].

Minimum Completion Time (MCT) [35] algorithm allocates a candidate task to the VM which results in lowest completion time for the task. MCT considers already assigned workload allocated for finding the appropriate VM for the task execution. MCT enhance resource utilization and reduces makespan as compared to RR and RS scheduling heuristics. However, MCT overloads the faster VMs and slower VMs remain idle which result in load imbalance [21]. Moreover, the performance of the machines degrades when more tasks are assigned to already over-loaded machines [32,43].

MaxMin [36] schedulers are based on MCT which assigns a task to the VM that promises minimum expected completion time for that task. MaxMin scheduling heuristic initially receive a list of unmapped tasks and VMs,

and completes the scheduling process in two steps: 1) finds the earliest finish time for a task using all VMs; 2) selects the task with a *maximum earliest finish time* for mapping to the concerned VM. On each scheduling step, the MaxMin removes the mapped task and updates VM ready time. The MaxMin approach penalize smaller jobs and favors large size jobs. Moreover, MaxMin based scheduling heuristics suffer from load imbalance issue for workload with a high number of large-sized tasks [21].
Authors in [42] have presented a modified PSO-based task scheduling and load balancing meta-heuristic algorithm. The proposed technique has used a novel inertia weight strategy to balance the exploration and exploitation mechanism of the particles. This results in achieving efficient task scheduling and load balancing. The presented technique is termed AdPSO and evaluated in terms of resource utilization, makespan, and throughput. However, the AdPSO technique is not deadline aware and task response time and task rejection ratio are not supported as scheduling objectives.

2.2 Batch dynamic scheduling heuristics

In Batch dynamic task scheduling a set of task is collectively mapped to VMs although the execution previous batch is not completed.
RALBA [21], a resource aware load-balancing algorithm provides a balanced distribution of workload according to the VMs computation capability. RALBA is a batch based task scheduler which assigns a batch of non-preemptive and independent tasks on fixed number of virtual machines. RALBA perform scheduling in 2 phases. In phase1, tasks are scheduled on VMs according to their computation power and computation requirement of the tasks. The second part of the RALBA scheduler assigns tasks to the virtual machines which executes that tasks earliest than others VMs. However, it suffers from the issue like inter-batch under-utilization of the resources, new batch formation based delay in processing, and unable to accommodate jobs with shorter deadlines.
In [29], the authors proposed an approach that uses a combination of *Chicken Swarm Optimization* (CSO) and *Improved Raven Roosting Optimization* (IRRO) algorithms. This approach uses strengths of both IRRO and CSO to provide balance in the local and global search which results in solving premature convergence, reduces the response time, execution time, and improves throughput. By using a hybrid algorithm (IRRO-CSO), a framework named ICDSF was presented for dynamic scheduling of independent tasks in a Cloud environment. The results show that the improvement in the execution time of the proposed approach and the existing heuristics is very small and the improvement in the response time is very high. However, this method does not take into account, the tasks deadlines and utilization of Cloud resource.

2.3 Dynamic scheduling heuristics

Mao et al. [23] has proposed the Max-Min based elastic task scheduling algorithm (ECMM) for load balancing. ECMM heuristic maintains VM and task status table which enable them to schedule task based on more real expected task completion time. VM status table shows the status of VMs which include tasks, VM_id, No of allocated tasks, execution time of tasks, and life cycle status, etc. Similarly, the task status table contains task_id, VM_id, task execution time, completion time of task, and latest update time. The incoming tasks are split into different batches based on time interval and allocate tasks to the VMs by employing MaxMin scheme without considering task migration and deadlines. Additionally, the employed approach is unable to achieve improved resource utilization, load balance, and minimized makespan.
TM-eFCFS: a task migration-based task scheduling heuristic proposed by Panwar and Negi in [27] that employs *First Come First Serve* heuristic as the base algorithm. To achieve faster execution and minimized the makespan, this approach uses non-live migration of tasks in the queue (waiting for the execution turn) or partially executed task to fastest idle VMs. The proposed approach comprises of two algorithms. 1) The algorithm selects incoming tasks and assigns them to the VM which provide *Earliest Completion Time* (ECT) and update VM ready time. The *second algorithm* performs task migration in which the unprocessed or partially processed tasks already assigned to the slower VMs are migrated to a faster machine (employing preemption mechanism). The first algorithm assign tasks based on early completion time and overloads faster machines. The idle or slower machines are not considered for the task-migration resulting in unbalanced distribution of workload that causes poor resource utilization [32].

Chen et al. [37] proposed a fuzzy control theory-based dynamic resource scheduling technique. This approach predicts the number of concurrent compute-resources required by the users using the historical information, i.e., a number of earlier requested resources, resource types, and the number of online users. The data center monitors the utilization of resources in real-time. This scheme ensure resources availability, efficiency, and prevent machine overload in peak hours. This scheduling technique depends on prediction model and feedback. However, such a prediction model is based on the historic resource scheduling information which is difficult to maintain.
In [25], authors have presented threshold oriented time-efficient, and dynamic task scheduling technique that targets to avoid tasking allocation to overloaded VMs [18]. In this approach, the incoming task is allocated to a virtual machine only when the already assigned workload of that virtual machine is less than a predefined threshold. The proposed algorithm performs task migration for assigning services to the task with higher priority. When tasks with short deadlines arrive, they are assigned to the fastest machine without considering the slower machines with minimum load. *Dynamic Load Balancing Algorithm* (DLBA) uses a threshold to check the VM for overloading; however, this approach ignores the under-utilized VMs, which reduce resource utilization and

increase load imbalance. Also, the task rejection ratio is higher because the approach does not accommodate jobs considering the deadlines.

A two-stage strategy has been proposed in [20] to minimize load imbalance and improve performance of task scheduling. At the first step, historical task scheduling information are used for task classification by using machine learning classifier. The use of historical scheduling information helps to create a specific number of VMs types in advance to save the time of VM creation at run-time. In stage 2, the dynamic task scheduler is presented for assigning the matched task to the corresponding virtual machine. The key contribution of this paper is to reduce the time by leasing virtual machines beforehand and employing the historical scheduling information. Moreover, the task requirements like execution cost and task deadline leading to the reduction in the task completion time and load of the VMs. Moreover, tasks response time, execution and penalty cost not considered for evaluation [17,32,43].

In [19], Mousavi et al. presented TLBO (*Teaching Learning Based Optimization*) and GWO (*Grey Wolves Optimization*) based hybrid algorithms. Social behavior and the hierarchical structure are modelled during hunting and used for the design of optimization algorithms. TLBO approach helps to investigate problem space, finding optimal parameters and settings for fulfilling the problem objectives. The results of presented scheme are evaluated against PSO, Biography Based Optimization (BBO) and GWO. Authors have claimed that the newly TLBO-GWO to the existing heuristics, especially for high volume data. However, for the tasks based on the deadline, the trend is not visible.

In [17], *Simulation-Based Optimization* (SBO) scheduling framework, named as RePro-Active has been proposed, which executes periodically. This approach solves issues like dependency on information on past activities and maintaining historical information. The algorithm starts from current conditions (rather than relying on the history data) and uses the SBO technique that tries to simulate possible prospective events to make better decisions. Although, it avoids dependency on the historical information; however, this approach results in low Average Resource Utilization Ratio (ARUR) as compared to Min-Min scheduling heuristic and load imbalance.

Heuristic-Based Load Balancing Algorithm (HBLBA) [18] employs configuration of host servers and tasks to VMs mapping. To reduce the waiting and completion time of the tasks, a queuing model has been adopted for task-to-VM mapping. The length of the host server queues is fixed and the queue length of VMs is dynamic. The proposed algorithm allows creating as many instances of hosts with the highest computation power as the data center allows. The slower hosts remain idle due to the unavailability of computing power required for a VM, which give rise to low resource utilization and load imbalance problems.

Kumar and Sharma in [24] proposed a deadline constrained dynamic scheduling algorithm which provides scalability by adding and removing VMs at run time. Virtual machines are leased based on the average number of the rejected tasks. When tasks are scheduled on the VMs then the task migration is performed from *Overloaded* VM (OVM) to *Under-loaded* VM (UVMs). Load

and capacity of each VM are calculated before task migration. The OVMs and UVMs are identified and sorted in descending and ascending order respectively. The threshold of the overloaded and underutilized VMs are taken from the existing approaches and fine-tuned by performing experiments. At the end of each interval, some of the under-loaded VMs are removed based on the average number of UVM. If the number of rejected tasks is high then the number of new VMs created is higher than required. The creation of unnecessary VMs increase scheduling overhead, load imbalance issue, and resources under-utilization issue. Furthermore, the task rejection ratio is high and the rejected tasks are not reconsidered for re-scheduling.

To reduce makespan and increase the number of tasks that meet their deadlines, [40] has proposed a flexible and elastic task scheduling algorithm in Cloud computing.The algorithm has the capability to automatically scale-up and scale-down based on the incoming requests from the service users. Architecture of the proposed approach comprises of components like controller node/scheduler, load analyzer, Elastic Load Balancer (ELB), and a component that perform provisioning and de-provisioning of Cloud resources.

In [48], an overall performance based resource aware dynamic scheduling algorithm for Cloud computing has been proposed. In this research, the combined effect of different evaluation parameters has been analysed. The idea is that most of time, improving one scheduling parameters can effect the performance of other.To compute the overall performance of Cloud task scheduler, the values of different evaluation parameters need to normalized. A novel normalization technique has also been presented.

In Summary, the literature study shows that the static and batch dynamic task scheduling schemes have issues like load-imbalance, poor resource utilization, and are unable to execute the newly arrived tasks with shorter deadlines. Moreover, the batch dynamic task scheduling algorithms suffer from the issues like delay in response time due to the formation of a new batch (results in higher response time) and poor resource utilization due to the formation of a new batch. A number of dynamic task scheduling approaches are batch-based, however, the majority of these algorithms are not deadline-aware and only a few of these approaches consider task deadlines. These approaches include DLBA [25] and DC-BLBA [24]; however, these approaches have high task-rejection ratio and imbalance mapping of jobs.

3 RADL Scheduling Scheme

This section discusses a detailed overview of the proposed task scheduling scheme that includes RADL's system architecture, performance model used, detail description of the algorithm, time complexity, and scheduling overhead analysis.

Table 1: Summary of the related work

Aproach	Strengths	Weaknesses
RS [33]	Simple implementation and lower scheduling overhead	Load imbalance and under-resource utilization
RR [34]	Minimal scheduling overhead and simple implementation	Not resource-aware, load imbalance and high makespan
MCT [35]	Improved Resource Utilization and makespan than RS and RR	Overload faster VMs, not support deadline based tasks
Max-Min [36]	Favors larger tasks by Allocating largest task to fastest VM,	Execution delay for smaller tasks [21], task deadline not supported
Dy-MaxMin [23]	Calculate the VMs status after a pre-defined interval, Real-time load balancing	Under-resource utilization [20, 21], Task deadline not considered
TM-eFCFS [27]	Task migration can play a role in balancing the load	No task migration for slower idle machines, minimized resource utilization, and load imbalance [20,38]
DLBA [25]	Threshold-based mapping, prevent over-provisioning of VMs	partial usage of threshold, Lower Resource utilization, high makespan and task rejection, and load imbalance
RALBA [21]	VMs share based distribution of workload	Reduced-resource utilization, High makespan, and task deadline not supported
TSSLB [20]	Reduced execution time by creating VM in advance	Dependent on historical information (previous scheduling) [17,18]
Repro-Active [17]	Not dependent on the historical information	Poor utilization of resources and load-imbalance [21], execute tasks in form of batches

3.1 RADL Background and System Overview

From the literature review, it has been observed that majority of the current task scheduling heuristics suffering from issues like poor resource utilization, load imbalance, high makespan, penalty, and task execution cost . Moreover, most of the these algorithms not support task deadlines and cost. RADL is a resource and deadline-aware load-balanced scheduling technique and composed of RADL Scheduler and LR-subScheduler. Figure 3 shows the basic architecture of the Cloud.

The user Infrastructure layer represents the Cloud interface used by the users who communicate with the Cloud through Internet. The cloud management layer comprises of several submodules like resource manager that determines

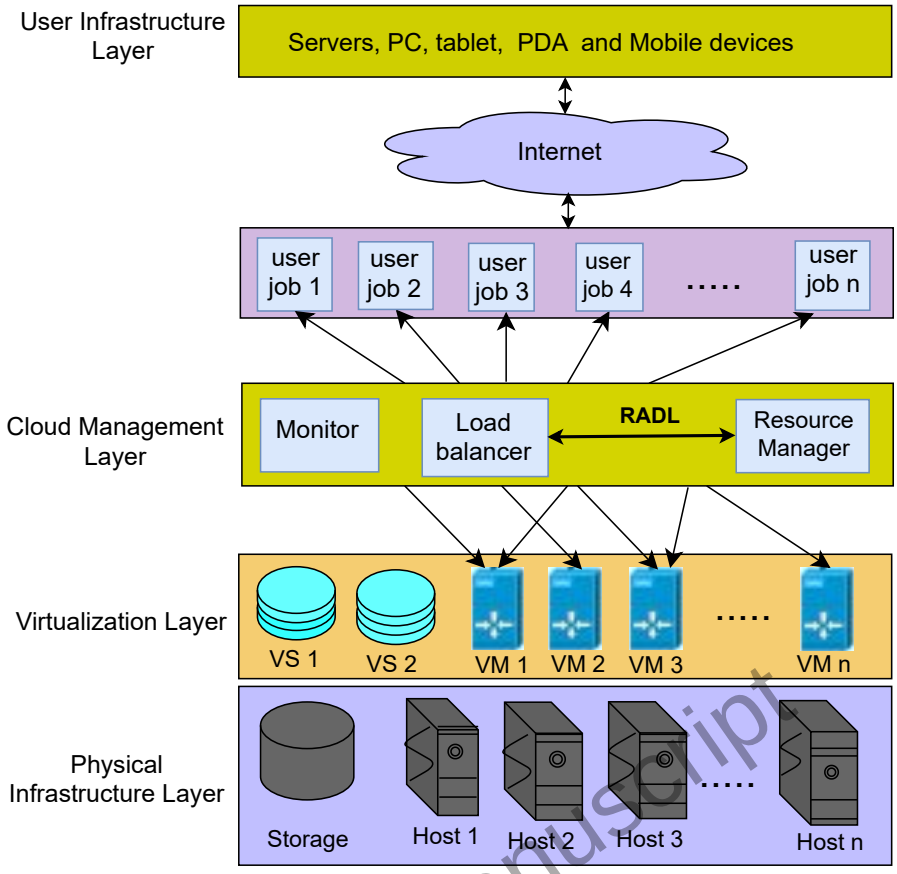


Fig. 3: Basic Architecture of Cloud

the user requirements in terms of computation resources, provide information of currently available resources, the monitoring module allows a system administrator to initiate and monitor activities of each layer, load-balancer manages the distribution of workload for execution among available virtual resources. RADL scheduling scheme is proposed at the Cloud management layer to enhance load balancing and improve utilization of Cloud resources. The virtualization layer represents virtual instances of the Cloud resources like virtual storage and virtual machines. The physical infrastructure layer represents the physical infrastructure of the Cloud.

To evaluate the performance of the proposed scheduling technique, experiment has been performed on Cloudsim [6] simulator. Datacenter manages hosts machines and the power of Datacenter is represented by the computation capability of host machines and servers on that Datacenter. One or more VMs are created on every host machine according to the VMs allocation policy defined by the *Cloud Service Provider* (CSP). RADL Cloud scheduling technique as-

signs task to VM based on just-in-time manner and the aims is to improve resource utilization, reduce task rejection, reduced task execution and penalty cost.

3.2 System Architecture of Proposed Scheme

RADL is a deadline-aware dynamic (just-in time based) task scheduling technique in Cloud computing that allocates incoming tasks in a load balanced manner. The Figure 4 shows the system architecture of the RADL model. Table 2 represents some of the notations used in RADL scheduling technique. RADL scheduling technique comprises of two heuristics i.e., RADL Scheduler and LR-subScheduler which consist of following major steps:
1. RADL scheduler accept task T_i with their size (sz_i) along with their deadline dT_i (in MilliSecond (MS). Task size is represented in Million Instructions (MIs). A set of VM (VMS) with computation capability of each VM is provided as an input parameter (shown in Figure 4, Algorithm 1). VMS represents the size of Cloud Datacenter.
2. The Completion Time (CT_{ij}) of the received task is calculated for all VMs and store in a storage structure (i.e., hashMap) (presented in Figure 4, Algorithm 1). CT_{ij} is the sum of the expected execution time of task T_i on VM_j and current load on the VM_j as shown in equation 2.
3. This step identifies VM_j which gives expected minimum completion time ($minCT_{ij}$) for T_i. $minCT_{ij}$ is compared with the deadline dT_i of the task T_i (shown in Figure 4, Algorithm 1).
4. This step will be executed based on the true value of condition at step 3. If $minCT_{ij}$ of task T_i is less than their deadline, task T_i is assigned to the VM_j that execute them in minimum time (Figure 4, Algorithm 1).
5. When task T_i is assigned to the VM_j, VM status and task status tables are updated (presented in Figure 4, Algorithm 1).
6. In case, $minCT_{ij}$ of task T_i is greater than their deadline dT_i, RADL Scheduler invokes LR-subScheduler. LR-subScheduler computes a suitable position in the task queue of VM_j for task T_i. Task T_i, VM_j, $minCT_{ij}$, and deadline dT_i are provided as input to the LR-subScheduler (shown in Figure 4, Algorithm 2). Based on the output (i.e., true or false) value of the LR-subScheduler, task T_i will be assigned to the VM_j or will select another VM from VMS which gives next $minCT_{ij}$ value for task T_i.

Table 2: Acronyms,notations and definitions used in RADL technique.

Notations	Description
Cloudlet	Notation that represent task in CloudSim simulator
VMS	List of Virtual Machines on a Cloud Datacenter
MI	Million Instructions (size of task)

MIPs	Computation capability of VM in Million Instructions Per Second
CT_{ij}	Task completion time on VM_j
$minCT_{ij}$	Smallest Time to complete task T_i execution on VM_j
dT_i	Task deadline in Millisecond (MS)
sz_i	Task size in MI
SPQ	Suitable Position in the task Queue of VM for current task
RTL	collection of tasks whose deadline not satisfied
pList	List of Parameters
tList	Collection of tasks
CT_c	Task Completion time that is selected for shifting
dT_c	Deadline of selected task
$newCT_c$	New completion time of task in case of shifting
$MindT_i$	Task in the VM task queue having shortest deadline
LR-subScheduler	Sub-scheduler that is invoked to identify SPQ for task T_i
Heterogeneous	VMs with high variation in their computation power

7. LR-subScheduler will check the tasks queue of VM_j to identify the task T_c, which has shorter deadline that is longer than the deadline of task T_i (shown in Figure 4, Algorithm 2). If such a task (i.e., candidate task T_c) is found, then new completion time of task T_c (i.e., $newCT_c$) and new completion time for task T_i (i.e., CT_i), will be computed. The $newCT_c$ and CT_i) will be compared with dT_c and dT_i respectively. In case any of these conditions become false then step 7 will be repeated until all the task queue of VM_j is scanned.
8. In case, both conditions at step 7 are true i.e., task T_i can be adjusted in the task queue of VM_j, then the positions of task T_i and T_c will be updated (shown in Figure 4, Algorithm 2).
9. In case, both conditions at step 7 become true and tasks positions are updated as represented in step 8. Task T_i will be mapped to VM_j, VM and task status tables are updated (shown in Figure 4, Algorithm 1),.
10. If condition at step 6 becomes false, a new VM that gives next minimum completion time will be identified (shown in Figure 4, Algorithm 1). In case, VM with next minimum completion time is found, then step 7 and 8 will be repeated.
11. when all VMs are scanned and task T_i is not scheduled in their deadline, then the task will be moved to list of rejected tasks. Similarly, this process of task scheduling will continue until a new task is available for scheduling (shown in Figure 4, Algorithm 1).

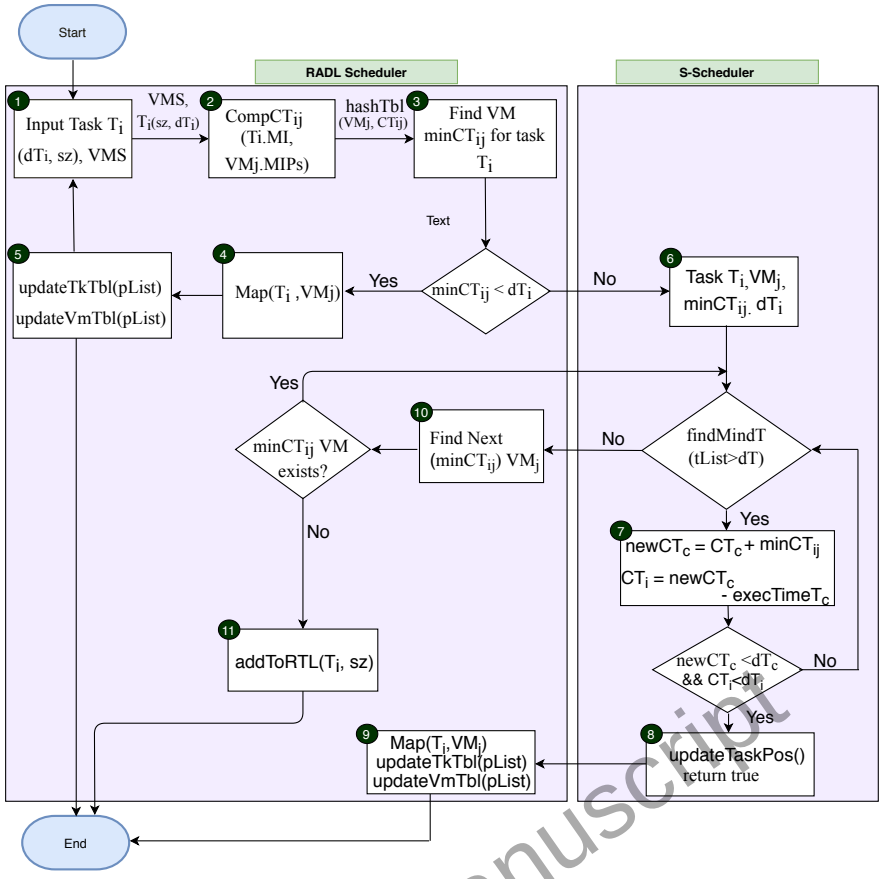


Fig. 4: RADL Scheduler

3.3 RADL System Model

Table 2 depicts acronyms, definitions, notations basic notations, and definitions used in the mathematical formulation of the proposed approach. To delineate the performance of RADL task scheduling technique, a consolidated system model for Cloud is devised [21]. Cloud datacenter comprises on a number of VMs (VMS) as presented in Equation 1 and a VM can be marked as VM_j.

$$VMS = \{VM_1, VM_2, VM_3, ...VM_m\} \tag{1}$$

where m represents the number of VMs in the VMS and $1 \leq j \leq m$. T_i shows the task to be mapped on VM_j that completes execution of T_i earliest than other VMs.

$$CT_{ij} = vmExecT_{ij} + vmRT_j \tag{2}$$

, Where (CT_{ij}) is the task T_i completion time of on VM_j that is the execution time of task T_i plus already assigned workload on that VM (depict in Equation

2). The execution time of the task T_i on VM_j is shown as $vmExecT_{ij}$ and $vmRT_j$ is the ready time (already assigned load) of VM_j. Equation (3) shows the computation of $vmExecT_{ij}$, which is the ratio of the size of task T_i (in Million Instructions (MI)) and the computation capability of VM_j (in MIPS).

$$vmExecT_{ij} = (\frac{size(T_i)}{VM_j.MIPS)}) \tag{3}$$

where the task size is represented in MI and MIPS represents the computation power of VM_j. The Equation (4) represents $vmRT_j$ and mathematically expressed as:

$$vmRT_j = \sum_{i=1}^{k} vmExecT_{ij} \tag{4}$$

where, k represent the total no of tasks allocated to VM_j. $minCT_{ij}$ (shown in Equation (5)) represents task T_i completion time of on VM_j that executes them in minimum time.

$$minCT_{ij} = min(CT_{ij}) \qquad \forall j \in 1, 2, 3, ...m \tag{5}$$

If the task T_i deadline is less than the T_i completion time of on VM_j which executes them in least amount of time, then a suitable place for T_i will be identified in the task queue of already mapped tasks to VM_j. SPQ represents Suitable Position in VM task Queue and return either true (1) or false (0) value (depicted in Equation (6)) and mathematically represented as:

$$SPQ = \begin{cases} 1\ SPQ \quad of \quad T_i \quad identified \\ 0\ otherwise \end{cases} \tag{6}$$

If SPQ returned value is 1 then the task T_i will be scheduled on VM_j, VM status will be updated.

$$VM\_CT_j = \sum_{i=0}^{k} (\frac{T_i.size(in\ MI)}{VM_j.MIPS}) \tag{7}$$

VM_CT_j represents VM_j completion time which is the summation of completion time of tasks scheduled to VM_j and expressed by Equation (7) [21]), where, k shows the no of tasks assigned to VM_j.The difference of task arrival time and service time is known as task response time, and mathematically represented by Equation (8).

$$RspTime_i = (execSTime_i - arrTime_i) \tag{8}$$

$RspTime_i$ is the response time of task T_i, $execSTime_i$ and $arrTime_i$ are the execution start time and arrival time of task T_i.

$$VMsCost = \sum_{j=1}^{m} VMCost_j \tag{9}$$

VMsCost is the sum of cost of all VMs used for the execution of users tasks (as shown in Equation 9). The VMCost can depends on multiple factors like VM computation power, number of CPUs, RAM, and storage requirement etc. In [31] authors have categorized VMCost based on their types. The VMs used in this article lies in category 1 of [31] and VMCost computed accordingly.

3.4 Performance Model of RADL Scheduler

The performance of RADL approach is evaluated by comparing with the existing state-of-the-art task scheduling approaches in terms of makespan, task rejection, response time, ARUR, penalty cost, and task execution cost. ARUR shows the average cloud resources utilization (as depicted in Equation (10) [17,21]) and is mathematically represented as:

$$ARUR = (\sum_{j=1}^{m}(VM\_CT_j)) \times \frac{1}{m \times Makespan} \tag{10}$$

ARUR values lie among 0 and 1, where, a lower value of ARUR shows poor utilization of cloud resources and higher value i.e., 1 represents better performance. The percentage of task rejection (%ageRT as shown in Equation (11) [41]) is achieved by dividing the rejected tasks by total tasks in the task list.

$$RT_{\%}age = (\frac{RTL.size() \times 100}{N}) \tag{11}$$

where, RTL is the rejected tasks list and N represent total tasks. Makespan represents the maximum time taken by a VM to complete execution of all the allocated tasks i.e., the time taken by a VM that finish the execution of tasks most recently. Mathematical representation of makespan is shown in Equation (12) [20,21,24,32,42].

$$Makespan = max(VM\_CT_j) \qquad \forall j \in 1,2,3,...m \tag{12}$$

where, m represent the number of VMs in VMS.

$$AvgRspT = \frac{1}{m}\sum_{j=1}^{m}(\frac{1}{k}\sum_{i=1}^{k}RspTime_{ij}) \tag{13}$$

Average Response Time (AvgRspT) (as shown in Equation 13) represent average response time of scheduled tasks. In equation 13, m shows the number of VMs in VMS and k shows the tasks that are allocated to every VM.

Cost parameters play a vital role in the selection and evaluation of Cloud task scheduling algorithm. The two important cost parameters include penalty cost and task execution cost [31]. Penalty cost is the cost incurred by the CSP by violating the Service Level Agreement (SLA). In our case, the CSP has

to pay back by not executing the user's tasks within their deadline i.e., task rejection. Penalty cost is the multiple of the number of rejected tasks and penalty rate. The penalty rate is the agreed penalty value between customer (Cloud user) and CSP on rejecting a task which can vary from provider to provider and customer to customer [30].

$$Penalty\_Cost = num\_RT * penalty\_Rate \tag{14}$$

num_RT is the number of task rejected. Minimized penalty cost represent the higher performance in terms minimum SLA violations and higher ratio of meeting tasks deadlines.

$$Task\_Exec\_Cost = VMsCost * Makespan \tag{15}$$

The execution cost of tasks is the multiple of cost of all VMs (VMsCost) and Makespan [31] as shown in Equation 15. Lower task execution cost shows better performance in term of reduced execution expenses for users jobs.

3.5 Proposed RADL heuristics

This Section describes proposed RADL task scheduling algorithm. RADL scheduling technique comprise of RADL scheduler and LS-subScheduler. The proposed RADL scheduler uses MCT based task-VM mapping and assign task to the VM if the expected completion of task is less than the their deadline. The proposed approach also update task and VM status in the task and VM status table. The LS-subScheduler locate fitted position in task queue of VM with MCT.

3.5.1 RADL Scheduler

To perform task to VM mapping, RADL Scheduler (Algorithm 1) accepts task T_i with their computation requiremetns (size) in the unit od Million Instructions (MIs), task deadline (dT_i) in Milli Seconds (MS), and set of VMs (VMS) with their processing capability as input. Task to VM mapping and list of rejected tasks (RTL) is returned as an output. The necessary initialization of the proposed algorithm is performed at lines 1-6 (Algorithm 1). The loop at lines 7–37 (Algorithm 1) will continue to iterate until all the available tasks are mapped to the VMs.

The For loop at lines 8-11 (Algorithm 1) compute the completion time of task T_i on every VM in the VMS using equation 2, 3, and 4 and store task completion time of each VM in a hashMap. The VM with MCT is identified (using equation 5) (lines 12, Algorithm 1). The method at (lines 12, Algorithm 1) takes hashMap as an argument that contains completion times of all VMs for task T_i. The completion time for task T_i is obtained (lines 13, Algorithm 1) and performed comparison with their deadline (line 14, Algorithm 1). If the decision at line 14 (Algorithm 1) is true, task is scheduled on the VM_j, VM

Algorithm 1: RADL Scheduler

```
   Input  : VM list with their computation power, Task T_i with size "sz_i" and
            deadline "dT_i"
   Output: Map (T_i, VM_j) i.e Mapping of Task T_i to VM_j
 1 CT_ij = 0
 2 minCT_ij = 0
 3 Vm VM = null
 4 RTL = null
 5 findSPQ = false
 6 dT_i = VmId = 0
 7 while T_i.exists() do
 8 |   for j = 1 to m do                                          } Step 2 and 3 shown
 9 |   |   CT_ij = computeCT_ij (T_i.MI, VM_j.MIPs)               } in Figure 4.
10 |   |   hashMap.add(CT_ij)                                     }
11 |   end                                                        }
12 |   VM_j = findVmWithMCT(hashMap)                              }
13 |   minCT_ij = VM_j.getKeyValue()                              }
14 |   if minCT_ij < dT_i then                                    } Step 4 and 5 shown
15 |   |   Map.add(T_i, VM_j)                                     } in Figure 4.
16 |   |   updateTkTbl(pList)                                     }
17 |   |   updateVmTbl(pList)                                     }
18 |   else
19 |   |   repeat
20 |   |   |   findSPQ = LR-subScheduler (T_i, VM_j, dT_i, minCT_ij) } call to the
21 |   |   |   if findSPQ == true then                            } S-Scheduler
22 |   |   |   |   Map.add(T_i, VM_j) updateTkTbl(pList)          } Step 9 shown
23 |   |   |   |   updateVmTbl(pList)                             } in Figure 4.
24 |   |   |   else
25 |   |   |   |   VM_j = findNVmWithMCT(hashMap)                 }
26 |   |   |   |   if VM_j != null then                           } Represent step 10
27 |   |   |   |   |                                              } of Figure 4.
28 |   |   |   |   |   minCT_ij = VM_j.getKeyValue()              }
29 |   |   |   |   end                                            }
30 |   |   |   end
31 |   |   until findSPQ == true or VM_j == null;
32 |   |   if findSPQ == false then                               } Represent step 11
33 |   |   |                                                      } of Figure 4.
34 |   |   |   RTL.add (T_i)                                      }
35 |   |   end                                                    }
36 |   end
37 end
```


and task status tables are updated (lines 15-17, Algorithm 1). On the false side of the decision, else part of the algorithm is computed (lines 18-36, Algorithm 1).

The LR-subScheduler is called that search the fitted location in the task queue of VM_j for task T_i (line 20, Algorithm 1). In the task queue of VM_j, the LR-subScheduler identifies task with minimum deadline greater than the deadline constraint of new unscheduled task (T_i) and if such a task is found in the task queue of VM_j then it returns a true value. If the returned value of LR-subScheduler is true (line 21, Algorithm 1), task T_i is assigned to VM_j,

the VM and task status tables are updated (lines 22-24, Algorithm 1). if the return value al line 21 (Algorithm 1) is false then the else block of statements are executed (lines 25 to 30, Algorithm 1) where next VM with $minCT_{ij}$ in the VM set is determined (line 26, Algorithm 1). The method at (line 26, Algorithm 1) finds next VM which completes the execution of task T_i in a least time for the remaining VMs. In case such VM is find then their $minCT_{ij}$ is acquired (line 28, Algorithm 1). The do-while loop is iterated until any of the conditions at line 31 (Algorithm 1) is true i.e., the task T_i is allocated to an appropriate VM or complete list of VMs is scanned. After scanning all the VMs, task T_i is not adjusted the then the task T_i is moved to the list of rejected tasks (line 34, Algorithm 1).

3.5.2 LR-subScheduler

The *LR-subScheduler*(Algorithm 2) is invoked by RADL scheduler (Algorithm 1) when the $minCT_{ij}$ of a task T_i is greater than the task deadline dT_i. The input parameters for *LR-subScheduler* comprise of task T_i with the deadline (dT_i), VM_j, and $minCT_{ij}$. The *LR-subScheduler* return true(1) value or false (0) value as an output. The necessary initialization of Algorithm 2 is represented in lines 1-5. The candidate task (T_c) which has the lowest deadline value in the task queue of VM_j that is greater than dT_i is identified (line 6, Algorithm 2). In case the candidate task is not found i.e., the condition becomes true (line 7, Algorithm 2), a false value is returned, and control move out of if condition (lines 7-8, Algorithm 2). On the false side of the if statement, the else block (line 9-37, Algorithm 2) will be executed. Execution time of T_i is computed (line 11, Algorithm 2). The new completion time of candidate task T_c (i.e., $newCT_c$) is identified by adding their completion time and execution time of task T_i (line 12, Algorithm 2). Similarly, execution time of candidate task T_c (i.e., $execTimeT_c$) and new completion time ($newCT_i$) of task T_i is obtained at (line 13-14, Algorithm 2). In case, the conditions at line 15 (algorithm 2) are false, the next task with a minimum deadline greater than dT_i is identified and the repeat-until loop (lines 10-20, Algorithm 2) will be executed again. The repeat-until loop is repeated until an appropriate location in the tasks queue of VM_j is determined or the complete queue is scanned and the appropriate location is not found. If the conditions at (line 15 of Algorithm 2) are true, the control moves out of the repeat-until loop and next statement (line 21 of Algorithm 2) after the loop is executed.
The position of the candidate task ($PosT_c$) is obtained from the task execution Map (TEMap)at (line 21 of Algorithm 2). The deadline dT_k, completion time CT_k), and updated completion ($newCT_k$) time of next task to T_c) which is termed as T_k) the task is obtained (lines 23-25 of Algorithm 2). If the condition at line 26 (Algorithm 2) becomes true then the flag value is set to zero and the control moves out of the For loop. if condition at (line 26 of Algorithm 2) becomes false then the For loop is executed again. The For loop is executed until the tasks in the queue are checked or the condition at line 26 (Algorithm 2) becomes true. If the condition at line 31 (Algorithm 2) becomes true

Algorithm 2: LR-subScheduler

```
   Input  : T_i with deadline dT_i, VM_j and minCT_ij
   Output: true or false
 1 T_c =null
 2 newCT_c = 0
 3 execTimeT_i =0
 4 CT_i = 0                                                    } Represent step 6
 5 execTimeT_c = 0                                             } of Figure 4.
 6 T_c = findMindT_i (tList) > dT_i                            }
 7 if T_c == null then
 8 |  return false
 9 else
10 |  repeat                                                   } Represent step 7
11 |  |  execTimeT_i = T_i.MI/VM_j.MIPs                        } of Figure 4.
12 |  |  newCT_c = CT_c + execTimeT_i                          }
13 |  |  execTimeT_c = TETMap(T_c)                             }
14 |  |  newCT_i = newCT_c - execTimeT_c                       }
15 |  |  if (newCT_c < dT_c && CT_i < dT_i) then               }
16 |  |  |  break:                                            }
17 |  |  else                                                  }
18 |  |  |  T_c = findNextMindT_i (tList) > dT_i               }
19 |  |  end                                                   }
20 |  until tList.hasNext();
21 |  PosT_c = getPosT_c(TEMap)
22 |  for k = PosT_c to vmQ.length do
23 |  |  dT_k = getDeadline(T_k)
24 |  |  CT_k = getCT_k (TEMap)
25 |  |  newCT_k = CT_k + minCT_ij
26 |  |  if (newCT_k > dT_k) then
27 |  |  |  flag = 0
28 |  |  |  break:
29 |  |  end
30 |  end
31 |  if (flag == 1) then                                     } Represent step 8
32 |  |  updateTaskPos()                                      } of Figure 4.
33 |  |  return true                                          }
34 |  else                                                    }
35 |  |  return false                                         }
36 |  end
37 end
```


then the position of the task is updated and a true value is returned. In case, condition at (line 31, Algorithm 2) becomes false, a false value is returned.

position of the task is updated, returned a true value and control moves out of the do-while loop (lines 16-20 of Algorithm 2). The do-while loop is iterated until an appropriate location in the tasks queue of VM_j is computed or the complete queue is scanned and appropriate location not found. If a fitted location for task T_i is not identified, the control is moved back to RADL scheduler (Algorithm 1) to pick another VM that has next minimum completion time.

Table 3: Complexity Analysis RADL algorithm

Algorithms	Dy-MaxMin[23]	PSSELB[17]	DLBA[25]	RALBA[21]	DC-DLBA[24]	RADL
Complexity	O(mN^2)	O(mN.n^2/2)	O(mN+N^2)	O(mN^2)	O(m^{2}N^2)	$O(mN + N^2) \approx O(N^2)$

Table 4: Computational overhead of RADL algorithm

Algorithms	RADL	PSSELB [17]	DLBA [25]	RALBA [21]	DC-DLBA [24]
Overhead(N times of Dy-MaxMin)	1.90	2.06	2.29	2.32	7.44

3.6 Complexity Analysis and Scheduling Overhead

To analyze complexity of RADL scheduling technique, we consider that N represents the total number of tasks and m represents the total number of VMs in a Cloud Datacenter. For a real Cloud Datacenter the following assertion has been made: N(the number of tasks)>>m (the number of VMs). RADL algorithm perform m number of comparison to finds VM that complete task execution in minimum time. The m x N becomes the total number of comparisons for N tasks. The RADL technique perform at most *m-1* matchings to identify VM that provide next lowest task completion time. To compute appropriate position for task, RADL algorithm invoke *LR-subScheduler* and the *LR-subScheduler* performs maximum N-1/m comparisons. The total number of comparisons become (m-1)(N-1)/m) to compute an appropriate location for the task in the task queue of the VM. The overall complexity of RADL technique becomes N(m+(m-1)N/m). Therefore, the time complexity of the RADL scheduler will become N(m+(m-1)N/m). Thus, the complexity of proposed scheduling technique has become $O(mN + N^2) \approx O(N^2)$. The RADL scheduling scheme has been analysed and evaluated in term scheduling overhead as compared to state-of-the-art task schedulers like Dy-MaxMin[23], PSSELB[17], DLBA[25], RALBA[21], and DC-DLBA[24]. Table 4 represent that *Dy-MaxMin* [23]) has lowest scheduling overhead than the rest of the approaches and is considered as baseline for comparison w.r.t to N times of *Dy-MaxMin*. The results shown in table 4 delineates that DC-DLBA[24] has the highest scheduling overhead. Its because this technique perform lease and release of VMs and task migration at run time. The scheduling overhead results shown int Table 4 reveals that the our proposed scheduling scheme RADL has outperform terms of overhead as compared to DC-DLBA[24], RALBA[21], DLBA[25], and PSSELB[17].

Table 5: Configuration of simulation environment

Simulation environment	Cloudsim 4.0 version
Experimental environment	Intel Core 2 Duo T6570, 4.00 GB Memory, 320 GB Hard drivez
Host Machines	30
Memory of every host machines	20000 MBs
The number of VMs	50 randomly generated VMs with different computation capacity (depicted in Figure 6) for GoCJ and Synthetic workload, and 16 heterogeneous VMs for HCSP based workload
Total Cloudlets	500, 600, 700, 900, 1000

4 Performance Evaluation

This section presents the experimental setup, workload generation performance evaluation, discussions of the RADL scheduling technique, and state-of-the-art task scheduling technique.

4.1 Experimental setup

To empirically evaluate the performance of the proposed approach, the Cloudsim [6] [22] simulator has been used. *Cloudsim* is well known java based open source tool that model and simulate the real Cloud environment and Cloud based services. The experimental environment includes CPU (Intel Core 2 Duo T6570 2 GHZ, 4.00 GB Memory, 320 GB Hard drive, eclipse IDE 2021 R and Cloudsim 4.0 (as presented in Table 5). We have extended the few of the classes of Cloudsim simulator that includes Cloudlet.java, Datacenter.java, and Broker.java. For our experimentations, a datacenter, 50 Virtual Machines, and 30 host machines have been used (presented in Table 5). The configuration of VMs used for all of our experiments shown in the Figure 6. In our experimentation, two realistic cloud/cluster traces and MapReduce logs of M45 supercomputing clusters based benchmark datasets have been used that includes Google like realistic workload that is GoCJ [32] [43] (as shown in Figure 5(b)), and Expected Time to Compute (ETC) model based on HCSP instances [43]. Additionally, synthetic workload based benchmark dataset from [21] has been used for evaluation.

A number of influential parameters has been used for evaluating and comparing the proposed scheduling algorithm. These evaluation parameters include makespan [51] [17][21][25][24], %age of task rejection[24], Mean ARUR [17][21] [23][51][49], and average response time [17][23][25][47]. However, most of the time improving one evaluation parameter may effect the performance of other i.e., executing tasks with shorter deadline before tasks with longer or no deadline can increase response time of tasks with longer deadline. This increases the importance to evaluate the overall performance of task scheduling algorithm as well. Overall performance represents the combined effect of the scheduling algorithm for more than one evaluation parameters (inversely related ones).

To compute the overall performance of tasks scheduling heuristics, the results of different parameters need to be normalized. In our previous work [48], we have presented a novel normalization technique especially designed for Cloud tasks scheduling which resolve the limitations of existing normalization techniques(as shown in equation 16 [48]).

$$V_n = 1 - \frac{V_o}{\sum_{i=0}^{k}(V_i)} \tag{16}$$

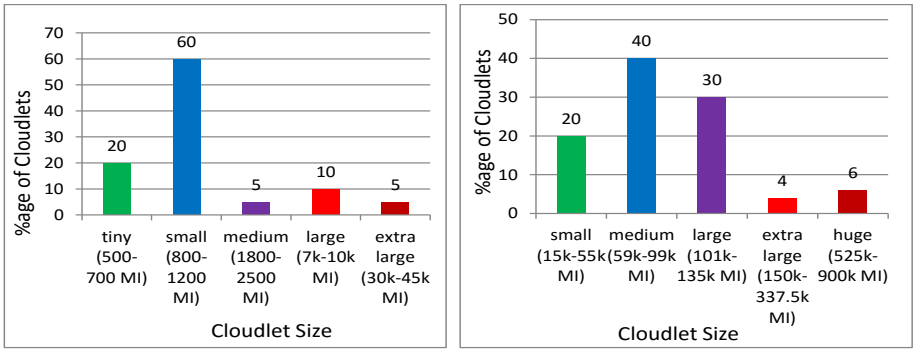
where V_n shows the normalized value, V_o represent the original number that need normalization, V_i represent the value of i^{th} parameter, and k shows the number of parameters.

$$OG = \left(\frac{\sum_{i=0}^{p}(P_i.Value)}{p}\right) \tag{17}$$

After normalizing the values of all parameters, overall performance in terms of overall performance gain(OG) is computed as depicted in equation 17. In equation 17 [48], where p shows the number of parameters and P_i value shows the normalized value of i^{th} parameter.

4.2 Workload formation

For experimentation, three benchmark datasets are used. However, these benchmark datasets does not contain deadline information which are added following the pattern used in [24]. The GoCJ [32][43][22][52] dataset is based on real world traces derived from traces of Google Cluster [45] and logs of MapReduce from M45 supercomputing cluster[44]. The GoCJ dataset using realistic high-performance computing Cloud tasks. The GoCJ dataset consists of tasks having different length i.e, 15000 MIs to 900000 MIs (as presented in Figure 5(b)) and is generated using a renown Monte-Carlo simulation method. To execute these tasks, virtual machines with different MIPS i.e., computation capacity are used. These MIPS lies in the range between 100-4000 MIPS. The HCSP [43,42] is a renown Expected Time to Compute model based benchmark dataset. This dataset approximate the actual behavior of heterogeneous Cloud environment [32]. There are a number of instances that include 1) HCSP instances with small size (16-64 VMs and 512-2048 tasks), medium size (128-256 VMs and 4096-8192 tasks), and large size (512-1024 VMs and 16384-32768 tasks). HCSP instances are categorized into i_hilo, c_hilo, s_hilo, i_lohi, c_lohi, and s_lohi where c shows consistent behaviour (uniform variations in task size and VMs MIPS), i for inconsistent behavior i.e., variation in tasks MI and VMs are not uniform, and s used for semi consistent behavior in term of variation in tasks MIs and VM MIPS. Moreover, lo represent low heterogeneity



(a) Synthetic Workload dataset (b) GOCJ dataset

Fig. 5: Cloudlets distribution for GoCJ[32] and Synthetic Workload dataset

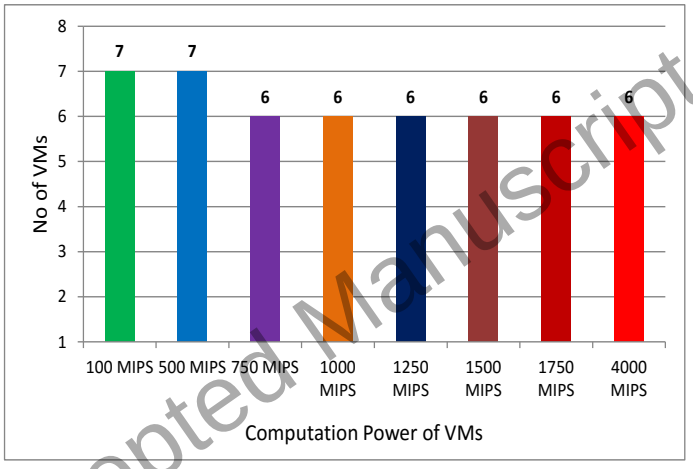
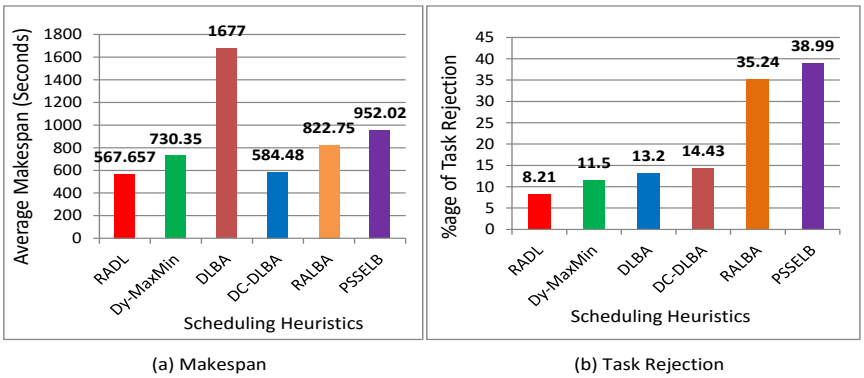


Fig. 6: VMs Computation power

and hi used for high heterogeneity. The virtual machine power ranges 19-7000 MIPS. In this article, HCSP instance with small size and lower complexity level has been used that comprise of i_hilo, c_hilo, i_lohi, and c_lohi. The synthetic benchmark dataset generated by [21] using monte-carlo technique. This dataset consists of tasks with various length ranges from 1 MI to 45000 MI (as presented in Figure 5(a)). The length of tiny tasks ranges from 1 to 250 MI, small from 800 to 1200 MI, medium from 1800 to 2500 MI, large from 7000 to



(a) Makespan (b) Task Rejection

Fig. 7: Makespan and Task Rejection results for GoCJ dataset

10000 MI, and Extra Large (XL) ranges from 30000 to 45000 MI.

4.3 Simulation Results

This Section shows simulation results of the proposed approach RADL and a comparison of these results with the existing state-of-the-art DC-DLBA [24], DLBA [25], Dynamic Max-Min (Dy-MaxMin) [23], PSSELB [17], and RALBA [21]. The performance evaluation metrics include makespan [17][21][25][24], %age of task rejection[24], Mean ARUR [17][21] [23], and average response time [17][23][25].

The first benchmark dataset used for evaluation of the proposed approach is GOCJ dataset proposed in [32] which comprise of large number of larger tasks as compared to small size tasks. Experimental results show that the proposed approach RADL has attained 28.66%, 195.42%, 2.96%, 44.94%, and 67.71% lower makespan (as presented in Figure 7(a)) and 40.07%, 60.78%, 75.76%, 329.23%, and 374.9% reduced task rejection (as shown in Figure 8(b)) for the execution of GoCJ benchmark dataset as compared to Dy-MaxMin [23], DLBA [25], DC-DLBA [24], RALBA [21], and PSSELB [17], respectively. Figure 9 presents experimental results in terms of penalty and task execution cost for the execution of GoCJ dataset. These results shows that RADL Cloud task scheduling scheme has achieved 374.72, 329.06, 75.69, 60.71, and 40.02% reduced penalty cost (as presented in Figure 9)(a) and 67.71, 44.94, 2.96, 195.42, and 28.66% lower task execution cost respectively against PSSELB, RALBA, DC-DLBA, DLBA, and Dy-MaxMin using GoCJ dataset respectively depicted in 9)(b). Figure 8(a) shows that RADL scheduling scheme has attained 60%, 56.67%, 6.67%, 23.33%, and 10% higher ARUR for the execution of GoCJ dataset against the Dy-MaxMin [23], DLBA [25], DC-DLBA [24], RALBA

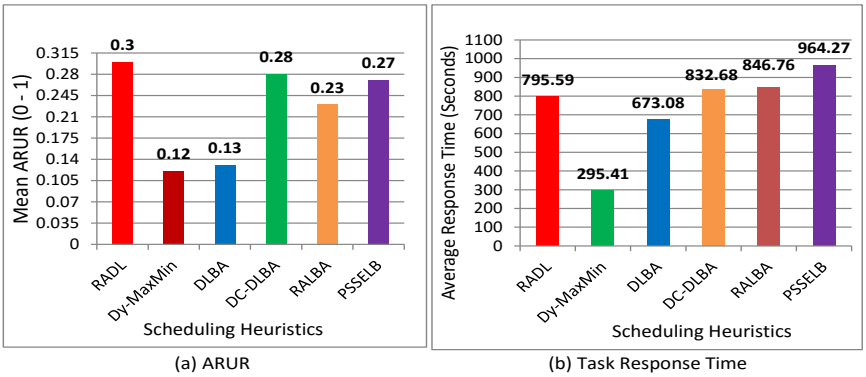


Fig. 8: ARUR and Task Response Time results using GoCJ datasets

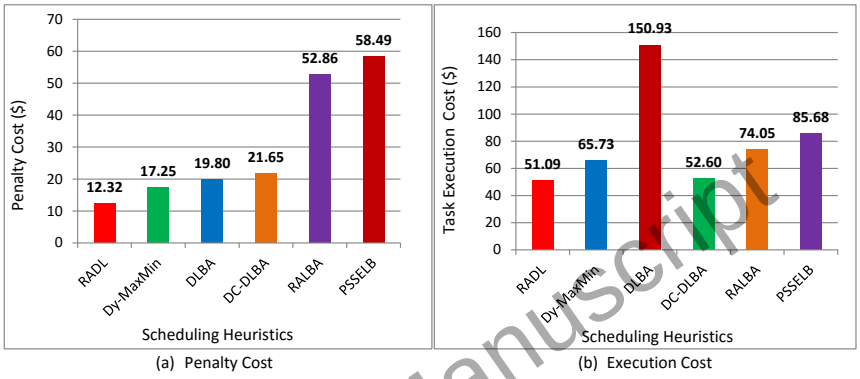


Fig. 9: Task Execution and Penalty Cost for GoCJ dataset

[21], and PSSELB [17], respectively. For the execution of GOCJ dataset, the RADL approach results in 4.66%, 6.43%, and 21.20% improved response time than DC-DLBA[24], RALBA [21], and PSSELB [17], respectively (as presented in Figure 8(b)).

In Cloud task scheduling, the overall performance of task scheduling heuristics cannot be ignored specially when we have contrasting performance parameters (i.e., increase of one aspect results in decrearse of other). This due to that most of the time improving one parameter can affect the performance of other parameters. To investigate the overall performance of proposed approach RADL, Overall Gain (OG) [48] of RADL scheduler has computed and compared with state-of-the-art Cloud task scheduling heuristics. For the execution of GOCJ dataset (as depicted in Figure 10), the proposed RADL approach has attain 4.68%, 13.70%, 3.28%, 12.37%, and 13.78% improved overall-performance gain against Dy-MaxMin [23], DLBA [25], DC-DLBA [24], RALBA [21], and PSSELB [17], respectively.

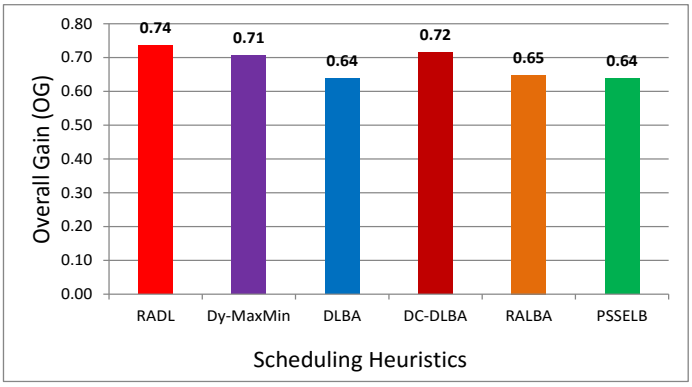
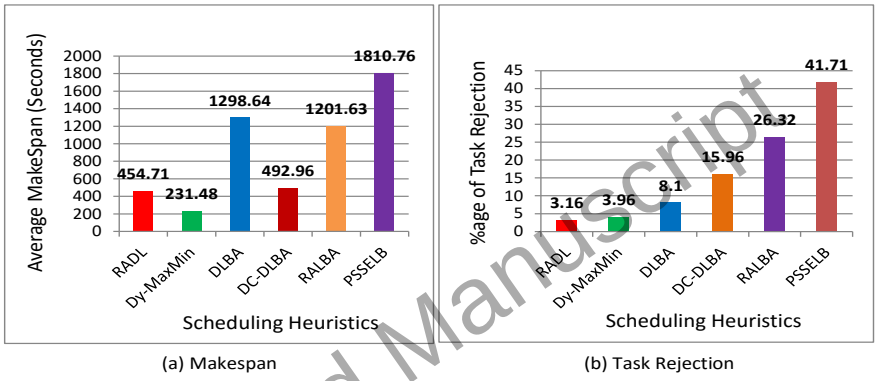


Fig. 10: Overall Performance Gain for GOCJ dataset



(a) Makespan (b) Task Rejection

Fig. 11: Makespan and Task Rejection results using HCSP dataset

Heterogeneous Computing Scheduling Problem (HCSP) [43,42] benchmark dataset consist of HCSP instances which uses ETC model. For the execution of HCSP dataset (as shown in Figure11(a)), the proposed scheduling heuristic RADL has attain -49, 185.59%, 8.41%, 164.26%, and 298.22% reduced makespan and 25.31%, 156.33%, 405.06%, 732.91%, and 1219.9% reduced task rejection as compared to the Dy-MaxMin[23], DLBA [25], DC-DLBA [24], RALBA [21] and PSSELB[17], respectively. Figure 12(a) shows that RADL heuristic has attained 55.88%, 41.17%, 61.76%, 11.76%, and 14.7% higher performance in terms of mean resource utilization for HCSP [43] dataset against Dy-MaxMin [23], DLBA [25], DC-DLBA [24], RALBA [21] and PSSELB [17], respectively. Figure 11(b) shows that RADL has achieved 17.4, 1, and 82.74%

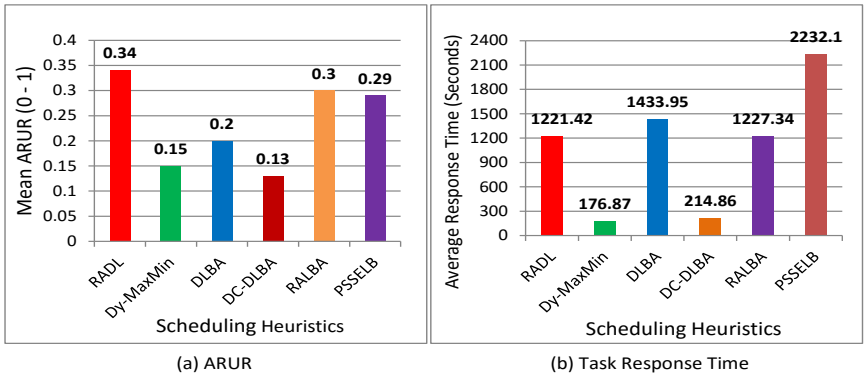


Fig. 12: ARUR and Task Response Time results using HCSP dataset

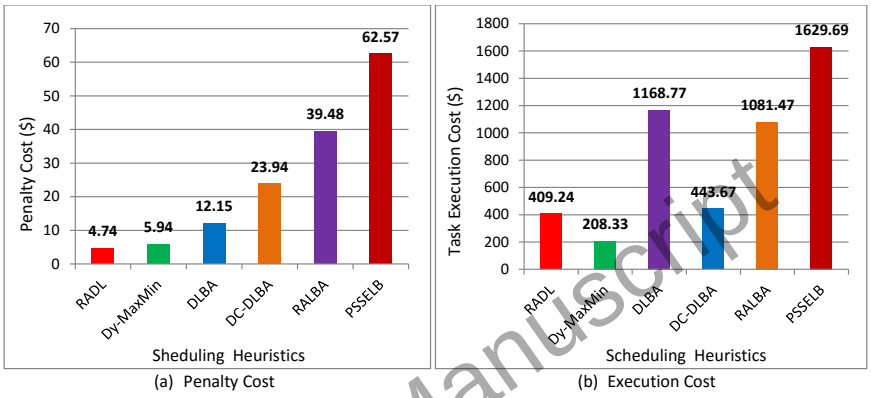


Fig. 13: Task Execution and Penalty Cost for HCSP dataset

improved task response time than DLBA[25], RALBA[21] and PSSELB[17] using HCSP benchmark dataset. The results depicted in Figure 13 shows that the RADL algorithm has gained 1219.94, 732.91, 405.06, 156.33, and 25.32% less penalty cost for the execution of HCSP dataset than PSSELB, RALBA, DC-DLBA, DLBA, and Dy-MaxMin respectively as depicted in Figure 13(a). Similarly, RADL has attained 298.22, 164.26, 8.41, and 185.60% minimized task execution cost than PSSELB, RALBA, DC-DLBA, and DLBA respectively as shown in Figure 13(b). Figure 14 shows that the RADL scheduling heuristic has attained 12.45%, 6.37%, 13.58%, and 27.74% improved overall performance for the execution of the HCSP dataset as compared to DLBA [25], DC-DLBA [24], RALBA [21] and PSSELB [17], respectively. Experimental results based on makespan using Synthetic workload [21] dataset is shown in Figure 15 (a). Synthetic workload dataset is a third benchmark dataset and positively skewed dataset where most of the tasks are smaller

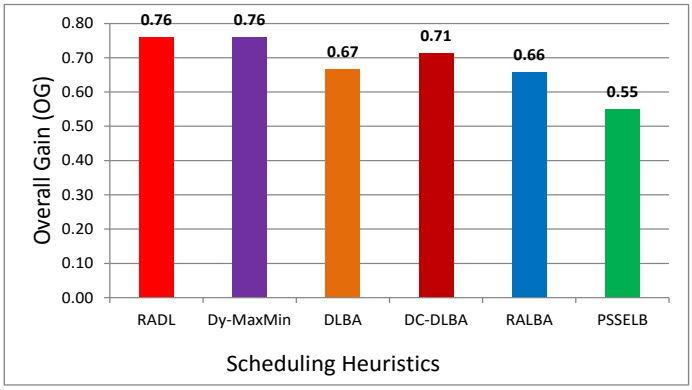
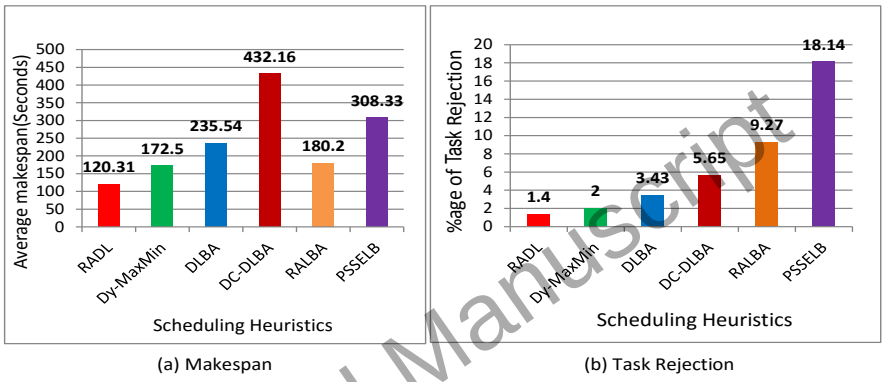


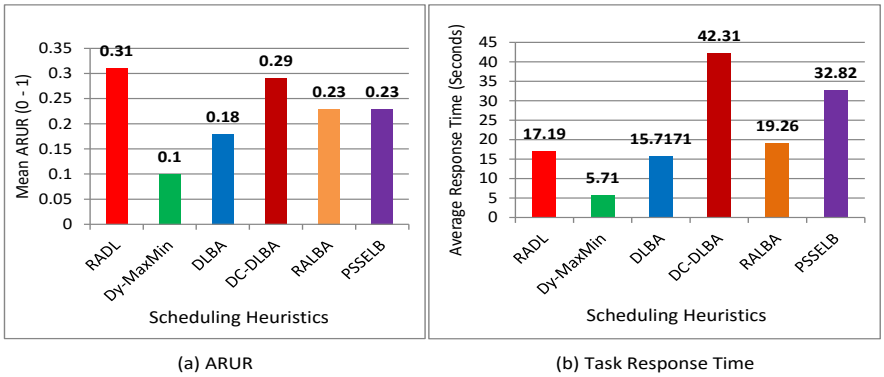
Fig. 14: Overall Performance Gain results using HCSP dataset



(a) Makespan (b) Task Rejection

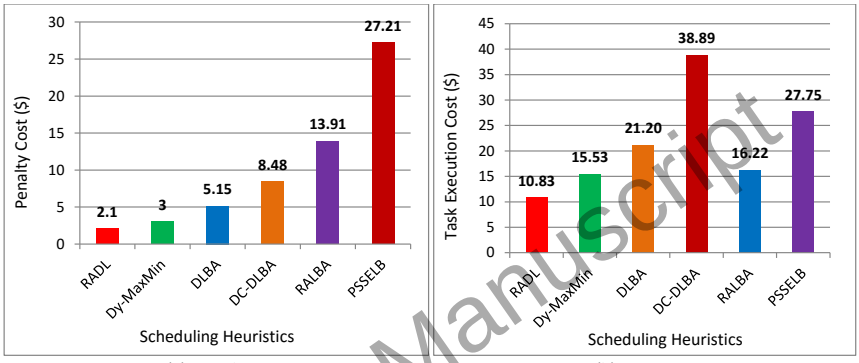
Fig. 15: Makespan and Task Rejection results for Synthetic dataset

in size and only few tasks are of larger size. Figure 15(a) reveals that the proposed approach RADL has attained 43.38, 95.78, 259.2, 49.78, and 156.28% lower makespan and 42.86, 145, 303.57, 562.14, and 1195.71% reduced task rejection (as shown in Figure 15(b)) using the Synthetic benchmark dataset against Dy-MaxMin [23], DLBA [25], DC-DLBA [24], RALBA [21] and PSSELB [17], respectively. Figure 16(a) shows that RADL has gain 67.74, 41.93, 6.45, 25.80, 25.80% higher performance in terms of mean resource-utilization for the Synthetic [21] dataset against Dy-MaxMin [23], DLBA [25], DC-DLBA [24], RALBA [21], and PSSELB [17], respectively. The results presented in Figure 16(b) reveal that the proposed RADL mechanism has attained 146.13%, 12.04%, and 90.92% improved task response time as compared to DC-DLBA



(a) ARUR (b) Task Response Time

Fig. 16: ARUR and Task Response Time results for Synthetic dataset based executions



(a) Penalty Cost (b) Execution Cost

Fig. 17: Task Execution and Penalty Cost for Synthetic dataset

[24] RALBA[21], and PSSELB [17], respectively for the Synthetic workload dataset. Experimental results based on penalty and task execution cost reveals that the RADL scheduling technique has attained 1195.71, 562.14, 303.57, 145, and 42.86% lower penalty cost (as depicted in Figure 17(b)), and 156.23, 49.75, 259.14, 95.74 and 43.35% reduced task execution cost than PSSELB, RALBA, DC-DLBA, DLBA, and Dy-MaxMin respectively for the execution of synthetic dataset.

Figure 18 presents that the RADL has gain 24.81%, 17.79%, 6.24%, 8.66%, and 11.41% improved overall performance gain as compared to PSSELB [17], DC-DLBA [24], Dy-MaxMin [23], DLBA [25], and RALBA [21], respectively on Sythetic workload based dataset.

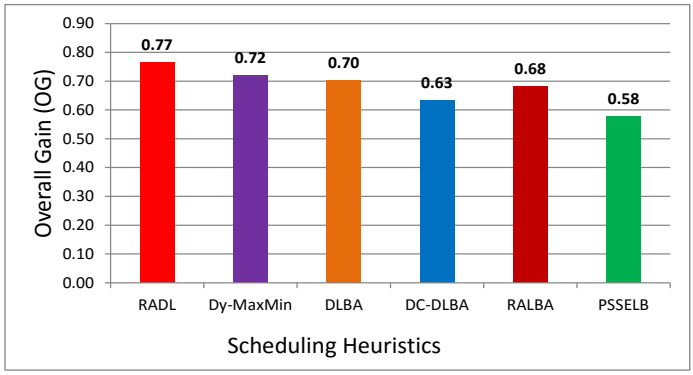


Fig. 18: Overall Performance Gain for Synthetic dataset

4.4 Results and Discussion

In-depth analysis of the experimental results reveals that the RADL scheduling technique has attained significant improvements in term of resource utilization, meeting task deadlines, reducing makespan, cost. It is because the RADL technique is deadline and resource aware. Experiments have been performed on three state-of-the-art benchmark datasets. The synthetic workload based benchmark datasets are a more positively skewed dataset. A positively skewed dataset contains a substantial number of small size tasks and few large size tasks, however, negatively skewed workload consists of a substantial number of longer tasks and reduced number of small size tasks. The GoCJ dataset used in the experimentation consists of relatively larger size tasks than that of synthetic dataset. Moreover, the HCSP instances based dataset employed in this experiments, consist of c_hilo, c_lohi, i_hilo, and i_lohi where c and i show consistency of task sizes and VMs MIPS where *c* shows consistent dataset and *i* is used for inconsistent workload, lo represent low heterogeneity and hi show high heterogeneity of the workload and resources. DC-DLBA achieved higher task response time, makespan, and task execution cost on synthetic dataset and reduced makespan and task execution cost for HCSP and GoCJ datasets. Its because, majority of the tasks in synthetic workload have small size and results in lower task rejection and creates few new virtual machines. The RADL scheduler has attained 259.2% improved makespan, 6.45% higher ARUR, 146.13% improved response time, 303.57% lower task rejection, and 259.14% lower task execution cost on synthetic dataset than DC-DLBA scheduler. Similarly, DC-DLBA attained reduced resource utilization and task execution cost on GoCJ and HCSP datasets as compared to synthetic workload. Dy-MaxMin attained reduced response time and task execution cost due to run-time updation of the task and VM status-tables result in the mapping of tasks on realistic expected completion time-based on VMs. However, re-

source utilization is low and task rejection is high too because it is neither creating new VMs nor giving priority to the tasks with shorter deadlines. The RADL attained 67.74%, 60%, and 55.88% higher resource-utilization than Dy-MaxMin on synthetic, GoCJ, and HCSP datasets, respectively. Also, RADL attains 43.38% and 42.86% improved makespan and task rejection, respectively for synthetic workload dataset. The scrutinized results show that PSSELB achieved lower resource utilization for synthetic workload due to the small number of larger tasks in the dataset. This is because the PSSELB selects the largest tasks possibly assign some of the larger tasks to the slower machine. The percentage of task rejection and penalty cost is high for all the three datasets because these do not consider deadline-based tasks. RADL has achieved 41.93%, 56.66%, 41.18% higher ARUR, 95.78%, 195.42%, 185.95% improved makespan, 145%, 60.78%, 156.33% reduced task rejection, 145%, 60.71%, and 156.33% reduced penalty cost, and 95.74%, 195.42%, and 185.6% less task execution cost as compared to the DLBA on synthetic workload, GoCJ, and HCSP datasets respectively. Experimental results show that the RALBA achieved improved performance in terms of makespan, and resource utilization as compared to PSSELB and DLBA. The scrutinized results show that the RADL attained 49.78%, 44.94%, and 164.26% improved makespan, 25.8%, 23.33%, and 11.76% improved resource utilization, 562.14%, 309.06%, and 732.91% reduced penalty cost as compared to RALBA on synthetic workload, GoCJ, and HCSP datasets, respectively. Its because RALBA scheduler is not deadline aware, which results in high task rejection and penalty cost. The overall in-depth analysis of the experimental results reveals that the RADL scheduler outperforms than its counterparts in terms of Resource utilization, meeting task deadlines, task response time, and makespan. However, RADL can not support workflow-based dynamic and SLA-aware task scheduling.

5 Conclusions, Limitations, and Future Work

Cloud computing has become an attractive platform for Cloud service providers and service users. To earn full advantage of Cloud and achieve high user satisfaction, Cloud service providers demand load balancing in terms of higher resource utilization and executing users' tasks within their deadlines. This leads to a need for efficient task scheduling algorithms. To attain enhanced utilization of Cloud resources and meeting task deadlines, several tasks scheduling algorithms has presented. However, the majority of the existing scheduling techniques are unable to achieve high utilization of Cloud resources and to meet task deadlines. In this research, a *Resource and deadlie Aware Dynamic Load-balancer for Cloud Tasks* (RADL) technique is proposed. The performance of the RADL scheduling technique is empirically evaluated and compared with state-of-the-art task scheduling schemes like RALBA, DC-DLBA, DLBA, Dy-MaxMin, and PSSELB using three scientific benchmark datasets. The evaluation results show that the RADL technique outperforms as compared to the state-of-the-art tasks scheduling algorithms in terms of ARUR

and in meeting the task deadlines, makespan, and task response time. The proposed approach reduces task rejection by maximizing the utilization of existing resources and adjusting the already mapped tasks. However, this may increase task rejection and task scheduling overhead if all the newly arrived tasks having shorter deadlines.

The RADL scheme assigns equal weight to every parameter like makespan, ARUR, task response time, and task rejection ratio. However, there is a possibility that users can give more importance to one evaluation parameter than others. The proposed approach assumes that the tasks should be computed intensively and independently and not consider the parameters like priority, bandwidth, communication latency, and memory. These parameters play an important role for workflow-based dependent tasks and need to consider for workflow-based task scheduling algorithms. The RADL scheduler accommodates the newly arrived deadline-based task, which can lead to starvation in a scenario where most of the majority of incoming tasks have a shorter deadline.

In the future, it is intended to extend the RADL scheduling technique for workflow-based (dependent tasks) dynamic task scheduling by considering priority, bandwidth, latency, and memory as scheduling objectives. Moreover, the RADL scheduling scheme can be used to assign weights to different evaluation parameters based on users' preferences. The proposed technique can be combined with the machine learning approach for VM load prediction.

References

1. Aruna, M., Bhanu, D., & Karthik, S. (2019). An improved load balanced metaheuristic scheduling in cloud. Cluster Computing, 22(5), 10873-10881.
2. Santos-Neto, E., Cirne, W., Brasileiro, F., & Lima, A. (2004, June). Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In Workshop on Job Scheduling Strategies for Parallel Processing (pp. 210-232). Springer, Berlin, Heidelberg.
3. Vaquero, L. M., Rodero-Merino, L., Caceres, J., & Lindner, M. (2009). A break in the clouds: towards a cloud definition., ACM SIGCOMM Computer Communication Review, 39(1),50–55.
4. Rodero-Merino, L., Vaquero, L. M., Gil, V., Galán, F., Fontán, J., Montero, R. S., & Llorente, I. M. (2010). From infrastructure delivery to service management in clouds. Future Generation Computer Systems, 26(8), 1226-1240.
5. Sharma, G., & Banga, P. (2013). Task aware switcher scheduling for batch mode mapping in computational grid environment. International Journal of Advanced Research in Computer Science and Software Engineering, 3(6), 1292-1299.
6. Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience, 41(1), 23-50.
7. S. Moore, R. van der M. Gartner, and Gartner,: Gartner Industry analyst firm. Gartner, Inc. (NYSE: IT).
8. Q. Zhang, L. Cheng, and R. Boutaba,: Cloud computing: State-of-the-art and research challenges, J. Internet Serv. Appl., vol. 1, no. 1, pp. 7–18, 2010.
9. Microsoft,: https://azure.microsoft.com. p. https://azure.microsoft.com.
10. S. B. Larry Page,: https://cloud.google.com. Google .
11. Amazon,: https://aws.amazon.com/. Amazon.

12. Nabi, S., & Khan, M. N. A. (2014). An Analysis of Application Level Security in Service Oriented Architecture. International Journal of Modern Education and Computer Science, 6(2), 27.
13. Nabi, S., Rehman, S. U., Fong, S., & Aziz, K. (2014). A model for implementing security at application level in service oriented architecture. Journal of Emerging Technologies in Web Intelligence, 6(1), 157-163.
14. Sajjad, A., Khan, A. A., & Aleem, M. (2018). Energy-Aware Cloud Computing Simulators: A State of the Art Survey. International Journal of Applied Mathematics Electronics and Computers, 6(2), 15-20.
15. Hazra, D., Roy, A., Midya, S., & Majumder, K. (2018). Distributed task scheduling in cloud platform: a survey. In Smart computing and informatics (pp. 183-191). Springer, Singapore.
16. Sulaiman, M., Halim, Z., Waqas, M., Aydın, D. (2021). A hybrid list-based task scheduling scheme for heterogeneous computing. The Journal of Supercomputing, 77(9), 10252-10288.
17. Alaei, N., & Safi-Esfahani, F. (2018). RePro-Active: a reactive–proactive scheduling method based on simulation in cloud computing. The Journal of Supercomputing, 74(2), 801-829.
18. Adhikari, M., & Amgoth, T. (2018). Heuristic-based load-balancing algorithm for IaaS cloud. Future Generation Computer Systems, 81, 156-165.
19. Mousavi, S., Mosavi, A., & Varkonyi-Koczy, A. R. (2017, September). A load balancing algorithm for resource allocation in cloud computing. In International conference on global research and education (pp. 289-296). Springer, Cham.
20. Zhang, P., & Zhou, M. (2017). Dynamic cloud task scheduling based on a two-stage strategy. IEEE Transactions on Automation Science and Engineering, 15(2), 772-783.
21. Hussain, A., Aleem, M., Khan, A., Iqbal, M. A., & Islam, M. A. (2018). RALBA: a computation-aware load balancing scheduler for cloud computing. Cluster Computing, 21(3), 1667-1680.
22. Hussain, A., Aleem, M., Iqbal, M. A., & Islam, M. A. (2019). Investigation of cloud scheduling algorithms for resource utilization using cloudsim. Computing and Informatics, 38(3), 525-554.
23. Mao, Y., Chen, X., & Li, X. (2014). 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). Springer, New Delhi.
24. Kumar, M., & Sharma, S. C. (2018). Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment. Computers & Electrical Engineering, 69, 395-411.
25. Mishra, S. K., Khan, M. A., Sahoo, B., Puthal, D., Obaidat, M. S., & Hsiao, K. F. (2017, July). Time efficient dynamic threshold-based load balancing technique for Cloud Computing. In 2017 International Conference on Computer, Information and Telecommunication Systems (CITS) (pp. 161-165). IEEE.
26. Deldari, A., Naghibzadeh, M., & Abrishami, S. (2017). CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud. The journal of Supercomputing, 73(2), 756-781.
27. Panwar, N., Negi, S., & Rauthan, M. M. S. (2017, October). Non-live task migration approach for scheduling in Cloud based applications. In International Conference on Next Generation Computing Technologies (pp. 124-137). Springer, Singapore.
28. Gutierrez-Garcia, J. O., & Ramirez-Nafarrate, A. (2015). Collaborative agents for distributed load management in cloud data centers using live migration of virtual machines. IEEE transactions on services computing, 8(6), 916-929.
29. Torabi, S., & Safi-Esfahani, F. (2018). A dynamic task scheduling framework based on chicken swarm and improved raven roosting optimization methods in cloud computing. The Journal of Supercomputing, 74(6), 2581-2626.
30. Nabi, S., & Ahmed, M. (2021). PSO-RDAL: particle swarm optimization-based resource- and deadline-aware dynamic load balancer for deadline constrained cloud tasks. The Journal of Supercomputing, 1-31.
31. Yazdanbakhsh, M., Isfahani, R. K. M., & Ramezanpour, M. (2020). MODE: A multi-objective strategy for dynamic task scheduling through elastic cloud resources. Majlesi Journal of Electrical Engineering, 14(2), 127-141.

32. Hussain, A., & Aleem, M. (2018). GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures. Data, 3(4), 38.
33. Xhafa, F., & Abraham, A. (2009, September). A compendium of heuristic methods for scheduling in computational grids. In International Conference on Intelligent Data Engineering and Automated Learning (pp. 751-758). Springer, Berlin, Heidelberg.
34. Rasmussen, R. V., & Trick, M. A. (2008). Round robin scheduling–a survey. European Journal of Operational Research, 188(3), 617-636.
35. Bardsiri, A. K., & Hashemi, S. M. (2012). A comparative study on seven static mapping heuristics for grid scheduling problem. International Journal of Software Engineering and Its Applications, 6(4), 247-256.
36. Elzeki, O. M., Rashad, M. Z., & Elsoud, M. A. (2012). Overview of scheduling tasks in distributed computing systems. International Journal of Soft Computing and Engineering, 2(3), 470-475.
37. Chen, Z., Zhu, Y., Di, Y., & Feng, S. (2015). A dynamic resource scheduling method based on fuzzy control theory in cloud environment. Journal of Control Science and Engineering, 2015.
38. Tabak, E. K., Cambazoglu, B. B., & Aykanat, C. (2013). Improving the performance of independenttask assignment heuristics minmin, maxmin and sufferage. IEEE Transactions on Parallel and Distributed Systems, 25(5), 1244-1256.
39. Kumar, M., Sharma, S. C., Goel, A., & Singh, S. P. (2019). A comprehensive survey for scheduling techniques in cloud computing. Journal of Network and Computer Applications, 143, 1-33.
40. Kumar, M., Dubey, K., & Sharma, S. C. (2018). Elastic and flexible deadline constraint load Balancing algorithm for Cloud Computing. Procedia Computer Science, 125, 717-724.
41. Kumar, M., & Sharma, S. C. (2018). PSO-COGENT: Cost and energy efficient scheduling in cloud environment with deadline constraint. Sustainable Computing: Informatics and Systems, 19, 147-164.
42. Nabi, S., Ahmad, M., Ibrahim, M., & Hamam, H. (2022). AdPSO: Adaptive PSO-Based Task Scheduling Approach for Cloud Computing. Sensors, 22(3), 920.
43. Hussain, A., Aleem, M., Islam, M. A., & Iqbal, M. A. (2018). A rigorous evaluation of state-of-the-art scheduling algorithms for cloud computing. IEEE Access, 6, 75033-75047.
44. Kavulya, S., Tan, J., Gandhi, R., & Narasimhan, P. (2010, May). An analysis of traces from a production mapreduce cluster. In 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (pp. 94-103). IEEE.
45. Chen, Y., Ganapathi, A. S., Griffith, R., & Katz, R. H. (2010). Analysis and lessons from a publicly available google cluster trace. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95, 94.
46. Ibrahim, M., Iqbal, M. A., Aleem, M., & Islam, M. A. (2018). SIM-cumulus: An academic cloud for the provisioning of network-simulation-as-a-service (NSaaS). IEEE Access, 6, 27313-27323.
47. Ibrahim, M., Nabi, S., Baz, A., Naveed, N., & Alhakami, H. (2020). Towards a task and resource aware task scheduling in cloud computing: An experimental comparative evaluation. International Journal of Networked and Distributed Computing, 8(3), 131-138.
48. Nabi, S., & Ahmed, M. (2021). OG-RADL: Overall performance-based resource-aware dynamic load-balancer for deadline constrained cloud tasks. The Journal of Supercomputing, 77(7), 7476-7508.
49. Ibrahim, M., Nabi, S., Hussain, R., Raza, M. S., Imran, M., Kazmi, S. A., & Hussain, F. (2020, May). A comparative analysis of task scheduling approaches in cloud computing. In 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID) (pp. 681-684). IEEE.
50. Sulaiman, M., Halim, Z., Lebbah, M., Waqas, M., & Tu, S. (2021). An evolutionary computing-based efficient hybrid task scheduling approach for heterogeneous computing environment. Journal of Grid Computing, 19(1), 1-31.
51. Ibrahim, M., Nabi, S., Baz, A., Alhakami, H., Raza, M. S., Hussain, A., & Djemame, K. (2020). An in-depth empirical investigation of state-of-the-art scheduling approaches for cloud computing. IEEE Access, 8, 128282-128294.
52. Nabi, S., Ibrahim, M., & Jimenez, J. M. (2021). DRALBA: Dynamic and Resource Aware Load Balanced Scheduling Approach for Cloud Computing. IEEE Access, 9, 61283-61297.