IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 6, 2021, accepted April 10, 2021, date of publication April 20, 2021, date of current version April 28, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3074145

DRALBA: Dynamic and Resource Aware Load Balanced Scheduling Approach for Cloud Computing

SAID NABI[1], MUHAMMAD IBRAHIM[2,3], AND JOSE M. JIMENEZ[4]

[1]Department of Computer Science, Virtual University of Pakistan–Rawalpindi, Rawalpindi 46300, Pakistan
[2]Department of Information Technology, The University of Haripur, Haripur 22620, Pakistan
[3]Department of Computer Engineering, Jeju National University, Jeju City 63243, South Korea
[4] Integrated Management Coastal Research Institute, Universitat Politècnica de València, 46022 València, Spain

Corresponding author: Muhammad Ibrahim (ibrahimmayar@uoh.edu.pk)

This work was supported in part by the Universitat Politecnica de Valencia through the Post-Doctoral PAID-10-20 Program.

ABSTRACT For the last few years, Cloud computing has been considered an attractive high-performance computing platform for individuals as well as organizations. The Cloud service providers (CSPs) are setting up data centers with high performance computing resources to accommodate the needs of Cloud users. The users are mainly interested in the response time, whereas the Cloud service providers are more concerned about the revenue generation. Concerning these requirements, the task scheduling for the users' applications in Cloud computing attained focus from the research community. Various task scheduling heuristics have been proposed that are available in the literature. However, the task scheduling problem is NP-hard in nature and thus finding optimal scheduling is always challenging. In this research, a resource-aware dynamic task scheduling approach is proposed and implemented. The simulation experiments have been performed on the Cloudsim simulation tool considering three renowned datasets, namely HCSP, GoCJ, and Synthetic workload. The obtained results of the proposed approach are then compared against RALBA, Dynamic MaxMin, DLBA, and PSSELB scheduling approaches concerning average resource utilization (ARUR), Makespan, Throughput, and average response time (ART). The *DRALBA* approach has revealed significant improvements in terms of attained ARUR, Throughput, and Makespan.This fact is endorsed by the average resource utilization results (i.e., 98 % for HCSP dataset, 75 % for Synthetic workload (improve ARUR by 72.00 %, 77.33 %, 78.67 %, and 13.33 % as compared to RALBA, Dynamic MaxMin, DLBA and PSSELB respectively), and 77 % for GoCJ (i.e., the second best attained ARUR)).

INDEX TERMS Cloud computing, task scheduling, dynamic task scheduling, scheduling heuristics, tasks to VM mapping, load imbalance.

I. INTRODUCTION

Cloud computing [1]–[4] has become an attractive and popular platform for individuals and organizations that provides the solution for execution and storage of large applications. Cloud computing work on a pay-as-you-use basis which satisfying the users' task scheduling requests in a scalable way [15] with the lowest cost than maintaining their computing infrastructure for their ad-hoc computing needs [5]. Cloud computing can be deployed as a public, private, hybrid Cloud [6], [7]. Cloud computing provides an opportunity to the organizations who maintain their own private Cloud, to use external Cloud resources in hours when users' demands are at their peak, instead of over-provisioning of their private infrastructure. The Cloud service providers need to efficiently manage the Cloud resources to improve the performance of users applications and enhance the utilization of the resources and with an optimal cost [8]. To reserve more resources can enhance the performance of the user application requirements; however, it will increase the resource usage cost. There is a need to have intelligent resource management schemes to satisfy the applications' performance requirements and efficiently utilize the available computing resources. The amount of work that a computing node needs to process the data is generally referred to as load or workload [9], [10]. In a real Cloud datacenter, the number of workloads

The associate editor coordinating the review of this manuscript and approving it for publication was Ali Kashif Bashir.

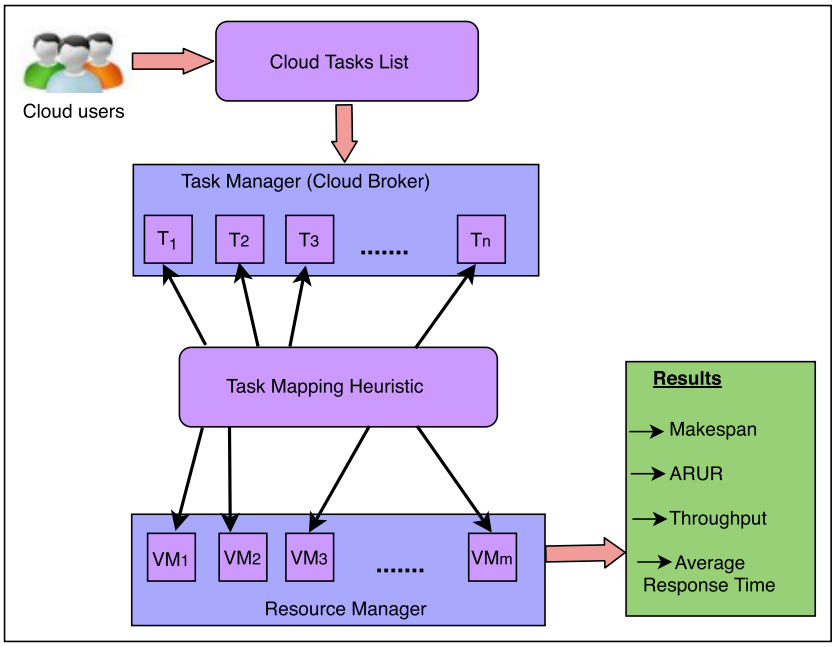


FIGURE 1. Task Scheduling Framework.

submitted to the Cloud for processing is much more than the number of computing nodes on the datacenter [8]. In such a scenario, the problem of resource assignments becomes more critical and challenging because it directly affects the performance of the Cloud system. Designing an efficient scheme for assigning resources to the workloads is known as scheduling strategy [11], face a lot of challenges. Scheduling is the process of decision making for mapping workloads (tasks) to Cloud computing resources to optimize some performance metrics like makespan, response time, resource utilization, and throughput [12]–[14]. Efficient task scheduling should guarantee the load balancing on the computing nodes. Load balancing algorithms play a vital role in improving the performance Cloud computing environment, especially in a heterogeneous environment [15]. Figure 1 shows the basic task scheduling framework of the Cloud environment. Cloud users are the customers that rent the Cloud services offered by the Cloud Service Providers (CSP) [16]. Cloud Tasks List is the workload that represents the actual requirements of the Cloud users. Task Manager or Cloud Broker accepts the customers' workload (i.e., Tasks list) and map these tasks on a suitable machine. The task mapping heuristic represents the proposed scheduling heuristics, which selects a task provided by the task manager and allocates them to a suitable VM which is provided by the resource manager [17]. The resource manager monitors the resource usage that includes both physical and virtual resources. When execution of customer workload is completed, then their results are obtained in terms of makespan, ARUR, Throughput, energy consumption, and task response time. Task scheduling schemes are categorized as static, batch dynamic and dynamic [10]. In static task scheduling approaches, the number of tasks with their computation requirements, number of Virtual Machines (VMs) and each VMs computation capacity should be known in advance [17]. Moreover, these task scheduling heuristics map all the tasks on the VMs before task execution starts. In static scheduling, once user tasks are assigned to the respective VMs, then the sequence of execution cannot be changed until the completion of execution of all tasks. The static scheduling heuristics are unable to handle any run-time changes during the execution of already mapped tasks. The static classical scheduling heuristics like Min-Min, Max-Min, and FIFO fails to utilize the resources efficiently and thus result in load imbalance [16] issues. Similarly, these approaches are unable to cope with scheduling based on the user priority. Batch dynamic task scheduling heuristics provide dynamism at the batch level. These techniques map a batch of tasks on a pre-defined number of VMs. Batch dynamic scheduling heuristics first statically maps all the tasks in a single batch before starting their execution and then assigns to VMs for execution [17]. In these scheduling techniques, the task scheduling strategy will remain the same for a particular batch of tasks and cannot be changed for the newly arrived batch of tasks. Batch dynamic scheduling heuristics suffer from issues such as 1) new batch formation based tasks processing delay, i.e., due to the arrival time differences of tasks in a single batch, the tasks which arrive early will have to wait till the completion of the new batch. This issue can also result in under-utilization of the resources when the tasks arrive in a non-uniform manner. 2) These algorithms can map the new batch without considering the load of the previous batch and can add new VMs for the newly arrived batch of tasks.

This issue can result in load imbalance, under-utilization of Cloud resources, and overload some VMs that can lead to performance degradation [18]. Moreover, these approaches fail to handle any run time problems during the execution of the task. They cannot support newly arrived high priority tasks in the later batches while VMs are already busy executing already assigned tasks. Batch dynamic heuristics are unable to re-estimate and recalculate the VMs' load during task execution. The tasks of later batches will wait for a long time to get their turn and increase overall execution time. An efficient and fair allocation of resources can achieve high user satisfaction and enhance the utilization of resources. Fair and efficient resource allocation can only be achieved through dynamic load balancing of local workload [19]. It can also help in avoiding bottlenecks and minimize resource utilization.

Dynamic [20], [21] task scheduling algorithms provide more flexibility as compared to the static and batch dynamic approaches. These heuristics can work with minimum information regarding resources and tasks. In the real Cloud environment, the workload is not fully known in advance, resources are heterogeneous [22], and the rate of tasks arrival fluctuates. Dynamic scheduling algorithms can schedule the tasks as it arrives, i.e., just in time basis without waiting for the formation of complete batch or execution of the already mapped tasks. These algorithms also can update the task status, VM load [19], [23]. Furthermore, dynamic task scheduling techniques can change the sequence of already mapped tasks, create new VMs, and migrate tasks during the execution of tasks. Therefore, it becomes imperative to use dynamic task scheduling and load balancing algorithms for the real Cloud environment [22].

In this research, a Resource Aware Dynamic Load Balancing Algorithm (DRALBA) has been proposed. DRALBA maps a set of independent, non-preemptive, and computationally expensive tasks on the available resources in a load-balanced way. It also updates the VMs' load and computation share at run time. The prime objective of the DRALBA approach is to improve resource utilization and task response time. The proposed approach calculates the computation share for each VM based on a set of tasks and then selecting a VM (with maximum computation share for assigning larger size task) whose length is less than or equal to the computation share of the selected VM. At each scheduling decision, the proposed scheduler updates each VMs load. The proposed approach update VMs load and computation share after several pre-defined iterations. In summary, our contributions in this research are as follows:

- An extensive literature analysis has been performed and identified limitations and strengths of state-of-the-art approaches.
- A thorough analysis of the static, batch dynamic, and dynamic state-of-the-art task scheduling has been carried out.
- We highlighted the optimization criteria and performance evaluation metrics for existing static, batch dynamic, and dynamic approaches.
- We have proposed a Dynamic Resource Aware Load Balancing Algorithm (DRALBA).

The rest of the paper is organized as: The discussion about the available contemporary scheduling approaches is presented in Section II. In Section III, the proposed task scheduling algorithm has been discussed. The discussion about the experimental setup and comparative analysis is delineated in Section IV. Section V contemplates the results and discussion, and finally, the conclusions and future work is presented in Section V.

II. RELATED WORK

This section of the article discusses and critically analyze some of the prominent state-of-art task scheduling heuristics. Fist-In-First-Out (FIFO) assign tasks to the VMs in the order in which tasks are received. FIFO [16] has simple implementation and less scheduling overhead. However, if a task with high computation requirements is submitted to the Virtual Machine (VM) before smaller tasks, then the smaller tasks have to wait for a long time. This issue negatively affects the user experience and increase response time for smaller tasks in the waiting queue.

Min-Min [24] scheduling heuristic selects the smallest task from the task list and assigns it to the VM that executes them in the minimum time as compared to other heuristics. Min-Min based scheduling algorithms have complex implementation and high overhead as compared to FIFO. Min-Min heuristic penalizes the larger tasks in terms of waiting time and favours smaller tasks. The min-Min approach fails to utilize Cloud resources efficiently [16]. Moreover, Min-Min heuristic cannot update VM load at run time.

Load Balanced Improved Min-Min (LBIMM) has been proposed in [25]. The LBIMM scheduling heuristic is an extended version of Min-Min scheduling heuristic. LIBMM can map task based on their priority (i.e., task execution time and cost.). Moreover, this approach allows Cloud users to select their required services. Authors have evaluated the proposed approach employing metrics like resource utilization, makespan, and average completion time of high priority tasks. However, this approach cannot update VM status at run time.

Max-Min [26] based task scheduling heuristics choose the largest task and assign it to the VM that execute that task in minimum time. This approach receives a list of tasks with their computation requirements, select the task with highest computation requirements, and compute its expected completion time on every VM. Max-Min map task on VMs with least expected completion time. Max-Min favours larger tasks and penalizes small size tasks in terms of response time. However, Max-Min fails to utilize the resources efficiently [16], [17]. Moreover, Max-Min approach also cannot update VM load at run time.

Authors in [19] have proposed Max-Min based dynamic task scheduling algorithm named as Dynamic Max-Min. This approach receives a list of tasks with their computation requirements and select the largest task from the received task

list and assign it to the VM with minimum completion time. This approach maintains the VM status and task status table. The proposed approach can update the VM status table and task status table at run time on an interval basis. This approach map tasks to VMs in more realistic expected completion time and reduce task response time. However, it suffers from the under-utilization of Cloud resources and load imbalance.

RePro-Active is reactive, proactive simulation-based dynamic task scheduling scheme proposed in [23]. This technique includes both reactive and proactive task scheduling algorithms. The reactive part of RePro-Active scheduling scheme uses a Round-Robin mechanism for the task to VM mapping. Proactive Simulation-based Scheduling and Load Balancing (PSSLB) is a simulation-based proactive algorithm. In RePro-Active scheduling technique, the incoming batch of tasks is divided into two halves. The first half of the tasks are scheduled using a polling method while another half of the tasks are mapped proactively. The PSSLB uses Simulate algorithm for prediction purpose and employs the principle of Max-Min scheduling heuristic. This algorithm selects the largest tasks and assigns to the VM that execute the task in minimum time. PSSLB performs better in term of Makespan and ARUR for some datasets. However, this approach cannot update VM status at run time.

Authors in [17] has proposed a Resource Aware Load-Balancing Algorithm (RALBA) in Cloud computing. RALBA is based on Max-Min task scheduling heuristic. This technique assigns tasks to VMs based on computation share of VMs. RALBA has two sub-schedulers including 1)Spill scheduler 2) Fill scheduler. Spill Scheduler select task with highest computation requirements that is less than or equal to the computation share of VM with highest share VM. Spill scheduler repeats until there is no task available whose computation requirement is less than the computation share of VM with highest computation share. Fill sub-scheduler assign the remaining un-mapped tasks the mechanism of Earliest Finish Time (EFT). RALBA has achieved better load balance and resource utilization. However, RALBA is unable to update VM Status at run time and also suffer from higher makespan and lower throughput.

Authors in [18] have proposed a threshold-based dynamic task scheduling and load-balancing technique. The threshold is the scaling limit (upper and lower) for machines used to identify overloaded and under-loaded VMs. The threshold is used to allow or disallow mapping of the newly arrived task to the VM. Moreover, the proposed approach migrate task from VM queue to the un-scheduled tasks list when a new high priority task arrives. For this purpose, a Dynamic Load Balancing Algorithm (DLBA) has been proposed. This approach prevents VMs from over-provisioning that may degrade their performance. However, DLBA does not consider under-utilized machines which can increase load imbalance and reduce utilization of Cloud resources. Moreover, this scheduling technique cannot update VM status at run time.

An Overall Gain based Resource Aware Dynamic Load-balancing scheduler (OG-RADL) has been proposed in [10]. The proposed scheduling technique computes the overall performance gain of Cloud by combining different evaluation parameters. These parameters include Average Resource Utilization Ratio (ARUR), Percentage of task rejection, tasks execution time, task response time. To combine the values of different evaluation parameters, there is a need to normalize these values. In OG-RADL, authors have also proposed a novel normalization technique that overcome the limitations of the state-of-the-art normalization techniques. However, the proposed approach not considered throughput as scheduling objective.

Authors have performed an experimental evaluation of eminent state-of-the-art static tasks scheduling heuristic in [27]. These algorithms have been critically investigated in terms of resource utilization, task execution time, throughput, and energy consumption. Moreover, individual load-imbalance is computed and compared. A well-known HCSP instances based benchmark dataset has been used for evaluation. However, dynamic task scheduling algorithms have not been considered.

Authors in [9] have proposed an adaptive threshold based task scheduling and load balancing approach in Cloud computing. This threshold is also termed as starvation threshold and named as starvation threshold-based load-balancing (STLB). The threshold is regularly updated based on the number of served requests and VM idle time. Moreover, to improve the Quality of Service (QoS) of Cloud systems, the proposed technique also consider the priority level of tasks. The objective of the STLB is to reduce task execution time, minimize VM idle time, and increase system stability in terms of minimizing the number of migrations. However, ARUR, task response time, and throughput not considered as evaluation parameter.

A Constraint Measure based Load Balancing (CMLB) technique has been proposed in [28]. At step 1, tasks are allocated to the virtual machines using round-robin scheduling technique capacity and a load of VMs is computed. In case, the load of the VM is greater than balanced threshold values then the load balancing technique is used for the task to VM mapping. This algorithm computes the deciding factor for every VMs and fill up the decision list and selection factor for every task and fill the selection list. After mapping the task to VM and the allocated task is removed from the form. The performance of the CMLB is evaluated using VM load and Capacity. However, the proposed approach does not consider ARUR, tasks response time, and throughput are not considered.

The authors in [29] have presented different Cloud scheduling algorithms. The objective of the proposed scheme is to improve data center processing time, task response time, and execution cost. The proposed approach considers two different aspects of Cloud scheduling. The first aspect focuses on improving individual scheduling objectives and comprises three different algorithms. These algorithms include:1) Cost Aware (CA) algorithm that reduces task execution cost, however, data center processing time and task response time

TABLE 1. Summary of the related work.

Aproach	Type	Strengths	Weaknesses
FIFO [16]	static	Simple implementation and minimal scheduling overhead	increased waiting time for smaller task and under-resource utilization
MinMin [24]	static	favor smaller task,lower makespan	penalize larger tasks under-resource utilization and load imbalance
LIBMM [25]	static	support task priority	cannot update VM status at run time
Max-Min [26]	static	Map largest jobs on the fastest VMs, Favors larger tasks	Execution delay for smaller tasks [17], cannot update VM status at run time
Dy-MaxMin [19]	dynamic	Update VM status after interval, Real-time load balancing	Under-resource utilization [17], [16], throughput not supported
Repro-Active [23]	dynamic	start execution with very few information in-hand	under-resource utilization and load imbalance [17], execute tasks in form of batches
RALBA [17]	batch dynamic	VMShare based balanced distribution of workload	High makespan and under-resource utilization, run time VM status update not supported
DLBA [18]	dynamic	prevent over-provisioning of VMs, consider task priority	partial usage of threshold, Lower Resource utilization, load imbalance, and high makespan
OG-RADL [10]	dynamic	improved overall Cloud performance and normalization technique	Throughput not supported
STLB [9]	dynamic	Reduce No of migrations and support task priority	ARUR, response time, throughput not considered
CMLB [28]	dynamic	prevent over provisioning of VMs	ARUR, response time, throughput not supported

are increased, 2) Load Aware (LA) scheduler improve task response time and datacenter processing time with increased in task execution cost, 3) Load Aware Over Cost (LAOC) algorithm provide a midway solution i.e., incurred lower execution cost than CA and higher execution cost than LA. Similarly, LAOC incurred reduced task response time and datacenter processing time than CA and higher data center processing time and task response time than LA. The second aspect of the proposed scheduling scheme provides a tradeoff solution to the Cloud task scheduling problem. For this purpose, the authors have presented a State-Based Load Balancing (SBLB) algorithm that reduces task response time and datacenter processing time without any increase in the task execution cost. However, the proposed approach does not consider resource utilization and datacenter throughput.

Multi-Objective optimization-based Dynamic task scheduling technique with Elastic resources (MODE) has been proposed in [30]. MODE is a scalable approach that allows the adding and removal VMs at run-time. A pre-defined criterion (threshold) is used to lease a new virtual machine or release the existing one. The performance of the proposed scheduling scheme is evaluated using the mean utilization of VMs, tasks execution time, total cost, and deadline violation. The proposed technique reduces the total cost and deadline violation. However, due to high scheduling overhead, MODE suffers from high makespan and under-utilization of Cloud resources. Moreover, the system throughput is not considered.

Authors in [31] have discussed the role of Cloud broker for Cloud data center selection and meeting the Cloud user's demands and quality of service requirements. Moreover, the authors have experimentally investigated state-of-the-art research articles related to Cloud brokering using response time and processing time. Experiments are performed in a simulation Cloud environment using seven (7) scenarios. Based on their analysis and findings, authors have suggested that to meet SLA-based user requirements, there is a need for efficient resource allocation techniques and complete architecture for the Cloud brokering mechanism. The authors have also recommended some research areas of Cloud brokering. However, the authors have not considered Cloud resource utilization and throughput for the evaluation of Cloud brokering algorithms. Moreover, the authors have not considered the dynamic behavior of Cloud user requests.

Authors in [34] has proposed a distributed Resource Aware Learning-as-a-Service (RALaaS) framework for learning based tasks. RALaaS comprise of Edge and Cloud based integrated resources. The proposed framework models the problem of resource allocation for learning demands. The aims is to increase accuracy and reduce resource requirement for learning based tasks.

In [35] authors has proposed an integrated context aware learning based channel selection framework that combine matching theory, lyapunov optimization, and machine learning technique. The proposed framework is highly important for reliable and efficient task delivery. The objective of the proposed approach is to maximize long term throughput, service reliability, and reduce energy consumption. To prove the guaranteed performance of the proposed framework, a rigorous analysis is provided. The performance of the proposed framework is verified in terms of service reliability and effectiveness using single Machine Typed Devices (MTDs) and multi machine typed devices scenarios.

Authors in [36] has presented a Multi-tier architecture in 5G environment named as H-CRAN. H-CRAN is an enhanced form of Cloud Radio Access Network (C-RAN). H-CRAN combines and enables various networks to work under a single controller. The proposed technique predicts

user expectations and choose the appropriate network based on users profile information. The Machine Learning (ML) module of the proposed technique is used to learn network constraints and user expectations under various payloads. ML paradigm provide intelligence in the network selection. The performance of the proposed framework is evaluated with real time traffic using simulation environment.

III. PROPOSED APPROACH

This Section presents a detailed discussion of the proposed Dynamic and Resource Aware Loading Balancing Algorithm (Dynamic-RALBA).

A. NOTATIONS, DEFINITIONS AND PROBLEM FORMULATION

This Section presents preliminary variables and notations, definitions, and problem formulation of the proposed dynamic RALBA.

1. *Makespan:* Makespan is the time required to complete the execution of the tasks on the mapped VMs in the Cloud datacenter, calculated in Eq. 1.

$$Makespan = Max\{CT_j\} = Max\{CT_1, CT_2, \ldots, CT_m\} \quad (1)$$

where m represent number of VMs and CT_j shows completion time of VM_j.

2. *ARUR:* The ARUR corresponds to average resource utilization by each of the heuristic during the complete execution of all the tasks on the available VMs.

$$ARUR = \frac{avgMakespan}{Makespan} \quad (2)$$

where avgMakespan is calculated in Eq. 3

$$avgMakespan = \frac{\sum_{j=1}^{m} Makespan_j}{m} \quad (3)$$

3. *Throughput:* The value of the throughput metric can be defined as the number of tasks executed per unit time as given in Eq. 4

$$Throughput = \frac{Total_{tasks}}{Makespan} \quad (4)$$

B. PROPOSED ARCHITECTURE

The proposed architecture of the Dynamic-RALBA is presented in Figure 2, which is based on the RALBA [17] approach. The proposed approach is dynamic because it can update the VM status (i.e., VM share) at run time. This updated status of VMs enables the proposed approach to assign the un-mapped tasks to the VMs in a more balanced manner. The proposed scheduling approach comprises of Dynamic-RALBA scheduler and a Dynamic-Updater sub-scheduler. Based on the task to VM mapping criteria, the Dynamic-RALBA scheduler is further divided into two parts. The first part of Dynamic-RALBA select VM with largest vmShare, identify the largest possible task for VM with largest vmShare, and assign that task to this VM.

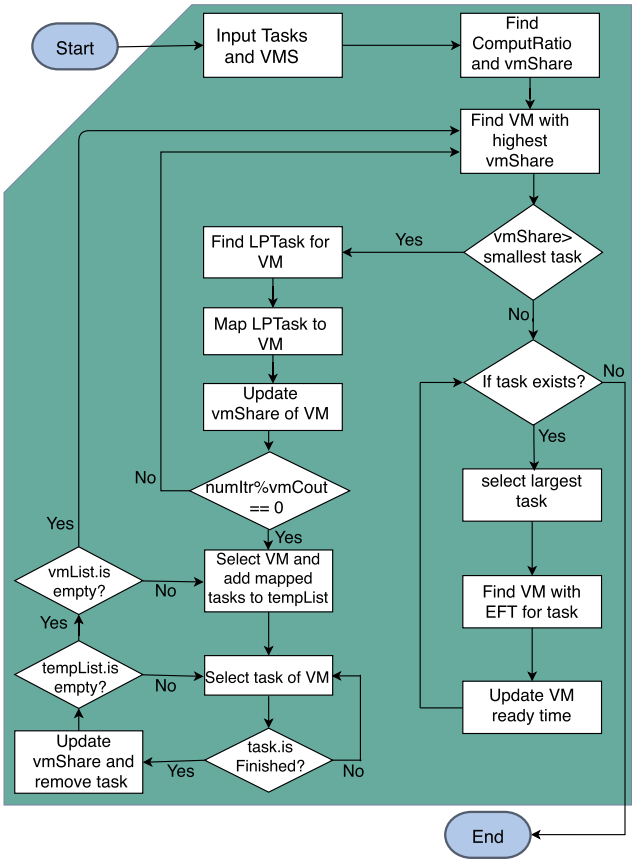


FIGURE 2. DRALBA System Architecture.

Dynamic-RALBA also updates vmShare of that VM accordingly. The second part of Dynamic-RALBA select the largest task and assigns it to the VM that finishes their execution earliest than other VMs. The Dynamic-Updater sub-scheduler update vmShare of every VM based on the tasks whose execution is completed. Table 2 presents preliminary variables and notations used in the proposed model and algorithm.

C. DYNAMIC-RALBA SCHEDULING ALGORITHM

This section describes the proposed task scheduling algorithm called Dynamic-RALBA or DRALBA. Dynamic-RALBA receives a set of VMs with their computation capabilities (in Millions of Instructions Per Second, i.e., MIPS) as vmMap and a set of independent and compute-intensive tasks (Cloudlets) with their computation requirements in Million Instructions(MI) as CloudletMap as input. vmMap and CloudletMap are converted into vmList and CloudletList (Line 1-2, Algorithm 1). Lines 3-6 of Algorithm 1 presents the necessary initialization. The total length of the tasks, i.e., task computation requirements in MI is computed VM (Line 7, Algorithm 1). Computation ratio (i.e.,vmComputeRatio) is computed dividing VM MIPS by total MIPS of the entire datacenter (Line 9, algorithm 1). Similarly, VM share (i.e.,vmShare) is identified for every VM by multiplying vmComputRatio with

TABLE 2. Preliminary variables and Notations used in DRALBA.

Variables/ Notations	Description
Cloudlet	Notation for a task in CloudSim
VMS	List of VMs in a Cloud data center
MIPS	Million Instructions Per Second (Computation Power of VM)
MI	Million Instructions (size of task/Cloudlet)
vmComputRatio	Computation ratio VMs
vmShare	Computing share of VM
LPTask	Largest possible task for VM
vmMap	Hashmap that stores set of VM in the form key value pairs
CloudletMap	Hashmap that stores set of Cloudlet
vmList	List of VMs on a datacenter
CloudletList	List of Cloudlets to be scheduled
vmShareMap	Computing share Map (vm, share) of VMs
vmTListMap	Task list Map (vm, taskList) of VMs
newVMShare	Updated computing share of VMs
vmTaskList	List that store mapped tasks of a VM
tempCltList	Temporary tasks list
EFT	Earliest finish time
totalLength	Sum of Cloudlet sizes in CloudletList

total computation requirements of received tasks and store them in vmShareMap(Line 10-11 Algorithm 1). The for loop (Line 8-12, Algorithm 1) is executed for each VM in the vmList. The while loop (Line 13-31, Algorithm 1) will continue to execute until the maximum vmShare is greater than the smallest task in the un-mapped tasks list. Line 14 (algorithm 1) identifies the VM with maximum vmShare. The largest task whose size (in MI) less than the vmShare of the selected VM is identified (Line 15, Algorithm 1) and if such task is found then if part of the statement is executed (Line 16-24, Algorithm 1). Counter variable numIterations is incremented by 1 (Line 17, Algorithm 1). The numIterations is incremented only when a task is mapped to the VM i.e., it counts the number of tasks mapped to the VM. In the if block of Algorithm 1, the task is mapped to the selected VM and vmShare of VM is updated by subtracting task MIs from their vmShare (Line 18-19, Algorithm 1). The updated vmShare is stored in the VM vmShareMap (Line 20, Algorithm 1). In Line 21 (Algorithm 1), vmTaskList is obtained, i.e., a list that stores those tasks that are assigned to a particular VM. The newly mapped task is added to vmTaskList, and the updated task list is added to vmTListMap (Line 22-23, Algorithm 1). The mapped task is removed from CloudletList (Line 24, Algorithm 1). In case, the largest possible task is not found for the selected VM, then break statement is executed, and control moves out of while loop (Line 26, Algorithm 1). If condition (Line 28, Algorithm 1) becomes true, then Dynamic-Updater scheduler is called (Line 29, Algorithm 1) to update VMs status. vmList.size()(Line 28, Algorithm 1) shows the total number of VMs on the datacenter. This work as a threshold for calling Dynamic-Updater sub-scheduler, which means that Dynamic-Updater sub-scheduler is called each time when the count of newly mapped tasks become equal to the total number of VMs. This threshold is made fixed after thorough fine-tuning by executing the proposed scheduler many times for different threshold values. When the first part of the Dynamic-RALBA scheduler ends then its second part (Line 32-39, Algorithm 1) start mapping of remaining tasks to VMs. This uses the Earliest Finish Time (EFT) based criteria for task mapping. Maximum task (Cloudlet) and VM with EFT is identified (Line 33-34, Algorithm 1) and the selected task is mapped to the VM that executes them in earliest than other (Line 35, Algorithm 1). vmShare of VM is computed, and the updated vmShare (newVmShare) is added to the vmShareMap (Line 36-37, Algorithm 1) and the mapped task is removed from CloudletList (Line 38, Algorithm 1). The while loop (Line 32-39, Algorithm 1) is executed until all the tasks are mapped to VMs, i.e., CloudletList becomes empty.

D. DYNAMIC-UPDATER SCHEDULER

This section delineates the Dynamic-Updater sub-scheduler (Algorithm 2) of the proposed scheduling approach. The inputs of the Dynamic-Updater sub-scheduler comprises of a set of VMs, list of tasks, vmShareMap, and vmTListMap whereas the output is task to VM map. Line 1-3 (Algorithm 2) presents the necessary initialization of the Dynamic-Updater sub-scheduler. For loop (Line 4-14, Algorithm 2) is executed for every VM in the vmList. The tempCltList temporarily stores the tasks that are assigned to VM_j (Line 5, Algorithm 2). The nested for loop (Line 6-11, Algorithm 2) repeats for each task in tempCltList, and every task is checked for their finishing (execution completion) using if statement (Line 7, Algorithm 2). In case the condition is true, i.e., execution of the task is completed (Line 7, Algorithm 2) then the completed task length (size in MI) is stored in cltTotalLength, and the completed task is removed from tempCltList (Line 8-9, Algorithm 2). This process is continued for each task in tempCltList (Line 6-11, Algorithm 2). After completion of the nested for loop, updated tempCltList is added to the vmTlistMap (Line 12, Algorithm 2) and vmShare of VM_j is updated and stored in the vmShareMap

Algorithm 1: Dynamic-RALBA Scheduler

Input : vmMap: set of VMs with their computation power,
CloudletMap: Tasks list with their computation requirements.
Output: Map (T_i, VM_j) i.e Mapping of Task T_i to VM_j

```
1  vmList = getVmList(vmMap)
2  CloudletList = getCloudletList(CloudletMap)
3  totalLength = 0, vmComputRatio = 0.0, vmShare = 0.0
4  newVmShare=0.0, numIterations = 0, vmTaskList = Null
5  vmShareMap<vm, vmShare> = Null
6  vmTListMap<vm, vmTaskList> = Null
7  totalLength = calcTotalLength(CloudletList)
8  for (Vm vm:vmList) do
9      vmComputRatio = vm.getMips/getTotalMips(vmList)
10     vmShare = totalCLength * vmComputRatio
11     vmShareMap.put(vm, vmShare)
12 end
13 while getMinCloudlet(CloudletList) ≤ getMaxShare(vmShareMap) do
14     vm = getMaxShareVm(vmShareMap);
15     Cloudlet= getMaxCloudlet4Vm (CloudletList, vm)
16     if Cloudlet != null then
17         numIterations++
18         bindCloudletToVm(Cloudlet.getCloudletId(), vm.getId())
19         newVmShare = vmShareMap.get(vm)-Cloudlet.getCloudletLength()
20         vmShareMap.put(vm, newVmShare)
21         vmTaskList = vmTListMap.get(vm)
22         vmTaskList.add(Cloudlet.getCloudletId())
23         vmTListMap.put(vm, vmTaskList)
24         CloudletList.remove(Cloudlet)
25     else
26         break;
27     end
28     if numIterations%vmList.size()== 0 then
29         updateCltVMTables(vmList, CloudletList, vmShareMap, vmTListMap)
30     end
31 end
32 while CloudletList.isEmpty() != true do
33     Cloudlet = getMaxCloudlet(CloudletList)
34     vm = getVmWithEFT(vmShareMap, Cloudlet)
35     bindCloudletToVm(Cloudlet.getCloudletId(), vm.getId())
36     newVmShare = vmShareMap.get(vm)-Cloudlet.getCloudletLength()
37     vmShareMap.put(v, newVmShare)
38     CloudletList.remove(Cloudlet)
39 end
```

Algorithm 2: Dynamic-Updater Scheduler

Input : vmList: set of VMs with their computation power,
CloudletList: Tasks list with their computation requirements.
vmShareMap, vmTListMap.
Output: Map (T_i, VM_j) i.e Mapping of Task T_i to VM_j

```
1  newVmShare = 0.0
2  tempCltList = Null
3  cltTotalLength = 0.0
4  for (int j = 0, j<vmList.size()) do
5      tempCltList = vmTListMap.get(j)
6      for (int k = 0, k<tempCltList.size()) do
7          if CloudletList.getById(CloudletList, tempCltList.indexOf(k)).isFinished()== true then
8              cltTotalLength = cltTotalLength + CloudletList.getById(CloudletList, tempCltList.indexOf(k)).getCloudletLength()
9              tempCltList.remove(k)
10         end
11     end
12     vmTListMap.put(vmList.get(j), tempCltList)
13     vmShareMap.put(vmList.get(j), vmShareMap.get(j) + cltTotalLength)
14 end
```

(Line 13, Algorithm 2). When tempCltList get empty, a new VM is selected. All the tasks that are assigned to newly selected VM are assigned to tempCltList (Line 5, Algorithm 2) and nested for loop (Line 6-11, Algorithm 2) is executed again.

E. COMPLEXITY AND OVERHEAD ANALYSIS

This Section discusses the complexity analysis of the proposed approach DRALBA against the state-of-the-art task scheduling approaches. For this purpose, we considered that N represents the total number of cloudlets and M shows the total number of Virtual Machines (VM) on the Cloud datacenter. In the real Cloud environment, the number of cloudlets is greater than that of VM (i.e., $N \gg M$). DRALBA perform M number of comparison to compute the computation share of VMs. In the worst case, DRALBA performs N number of comparison to minimum cloudlet in the cloudletlist. To identify VM with maximum share, DRALBA performs M number of comparisons i.e., M+MN. The dynamic updater scheduler performs MN number of comparisons to update the status of every VM. Therefore, the complexity of the DRALBA scheduler will become M+MN(MN). The overall complexity of the DRALBA becomes $O(M^2N^2)$ The complexity of scheduling algorithms is one of the important factors. However, reducing complexity is not the scheduling objective of this article. It is because the dynamic approach somehow increase scheduling complexity compared to static counterparts and this little

TABLE 3. Computational complexity of scheduling algorithms.

Algorithms	Dy-MaxMin	PSSELB	DLBA	RALBA	DRALBA
Complexity	$O(MN^2)$	$O(MN.N^2/2)$	$O(N^2)$	$O(MN^2)$	$O(M^2N^2)$

increase in the complexity is considered as a cost for achieving higher resource utilization and load balancing.

IV. EXPERIMENTAL SETUP

Various approaches, including analytical, experimental, and simulation are considered for the performance evaluation of any Cloud scheduling heuristics. The problem concerning the experimental approaches on the real Cloud is that they are costly and sometimes requires an expert to configure the environment. Secondly, performing simulations on the Cloud platform involves high monitory cost. The problem with the analytical approaches is that they lack in evaluating the proposed scheduling heuristics thoroughly. The simulation techniques are widely used for performance investigation of the contemporary approaches. In this research, we have considered Cloudsim [7] for an in-depth investigation of the proposed approach against the available four contemporary approaches (RALBA, Dynamic MaxMin, DLBA, and PSSELB). The simulations were executed on a machine equipped with Intel Core i5-8500 (TM) CPU (clock speed of 3.00 GHz) and 20 GBs of RAM.

To investigate the performance of the available scheduling approaches, three different benchmark datasets namely the HCSP (Heterogeneous Computing Scheduling Problems) dataset [12], [13], Google like dataset called GoCJ [33], and Synthetic workload [17] have been considered.

The HCSP instances-based benchmark dataset is based on the Expected Time to Compute (ETC) model and is generated in such a way that can approximate the real behavior of a heterogeneous computing environment. HCSP instances are divided into three levels based on their sizes and complexities. These categories include small size instances (512-1024 heterogeneous tasks and 16 to 64 heterogeneous VMs), medium-size HCSP instances comprised of 4096 to 8192 heterogeneous tasks, and 128 to 256 heterogeneous VMs. Similarly, large size instance include 16384 to 32768 tasks and 512 to 1024 VMs. In this paper, HCSP instances with a small size dataset which have 1024 heterogeneous tasks and 32 heterogeneous VMs are used. HCSP instances use the C-THMH pattern, where C shows consistency level which includes inconsistent (i), consistent (c), and semi-consistent (s). Moreover, TH represents task heterogeneity and MH stands for machine heterogeneity. In this paper, four different instances that are c-hilo, i-hilo, c-lohi, and i-lohi are used. Each of the instance having 1024 tasks and 32 VMs.

Synthetic workload based benchmark dataset published in [17], comprise of tasks with heterogeneous sizes ranges from 1 MI to 45000 MI. These tasks include tiny size (1-250 MI), small (800-1200 MI), medium (1800-2500 MI), large (7000, 10,000 MI), and extra-large (30,000 to 45,000 MI). The number of tasks consist of 500, 600, 700, 800, 900, 1000 and are randomly generated using a well-known Monte-Carlo simulation method [32].

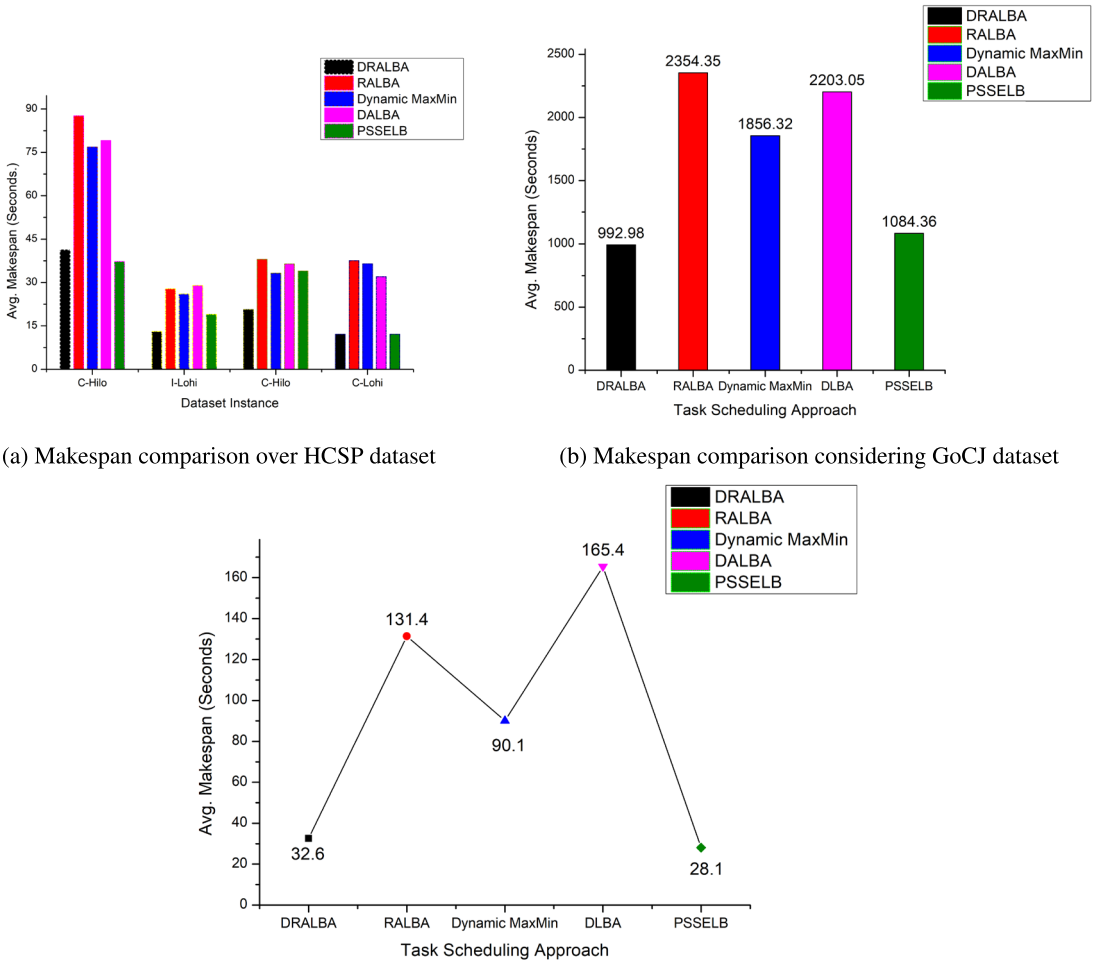
GoCJ benchmark dataset published in [33] is a google like realistic workload generated from realistic google cluster traces using a well-known Monte-Carlo simulation method [32]. The sizes of tasks in the GoCJ dataset range from 15000 (15k) to 900k and include small size tasks (15k-55k MI), medium size tasks (59k-99k MI), large size tasks (101k-135k MI), extra-large size (150k-337.5k MI), huge size tasks (525k-900k MI). For the execution of the Synthetic workload and GoCJ based benchmark dataset, 50 heterogeneous (in terms of computation power) are used. The computation of these VMs ranges from 100 MIPs to 7000 MIPS.

A. EXPERIMENTAL EVALUATION

Various performance metrics, including Makespan, throughput, average resource utilization (ARUR), SLA violation, energy consumption etc., are considered for performance investigation. The Cloud end-users are always concerned about the response time with minimum SLA violation, whereas the Cloud service providers are more concerned about the revenue generations. We have utilized ARUR, Throughput, Makespan, and ART to investigate the performance of our proposed approach against the available contemporary task scheduling approaches. The obtained results and the discussion is delineated below.

Figure 3 plots and explains the results regarding the Makespan attained for all the presented approaches two renowned and large-scale datasets such as HCSP dataset instances (i.e., C-hilo, C-lohi, I-hilo, and I-lohi) and GoCJ and on Synthetic workload. The results have shown that the proposed D-RALBA approach dominated the rest of the approaches concerning the Makespan on the C-Hilo dataset instance. Figure 3a shows that DRALBA consumes 54.82 %, 50.16 %, 52.06 %, and 14.90 % less execution time (for the HCSP dataset instances) as compared to the RALBA, Dynamic MaxMin, DLBA, and PSSELB respectively. For the C-Lohi dataset, the PSSELB showed 10 % improvement as compared to the proposed approach and also has been able to lead the rest of the two compared approaches. Dynamic MaxMin and PSSELB have shown moderate performance behaviour. The RALBA approach shown a very poor Makespan performance on the C-hilo dataset as compared to the rest of the approaches; however, behaviour similar to Dynamic MaxMin and DLBA is observed for the other three dataset instances.

On the Lohi dataset, most of the tasks are of smaller size and the heterogeneity of resources is lower as compared to Hilo dataset instances. The proposed and PSSELB have shown almost similar behaviour for the C-Hilo dataset instance.



(a) Makespan comparison over HCSP dataset

(b) Makespan comparison considering GoCJ dataset

(c) Makespan comparison using Synthetic workload

FIGURE 3. Makespan comparison on various datasets.

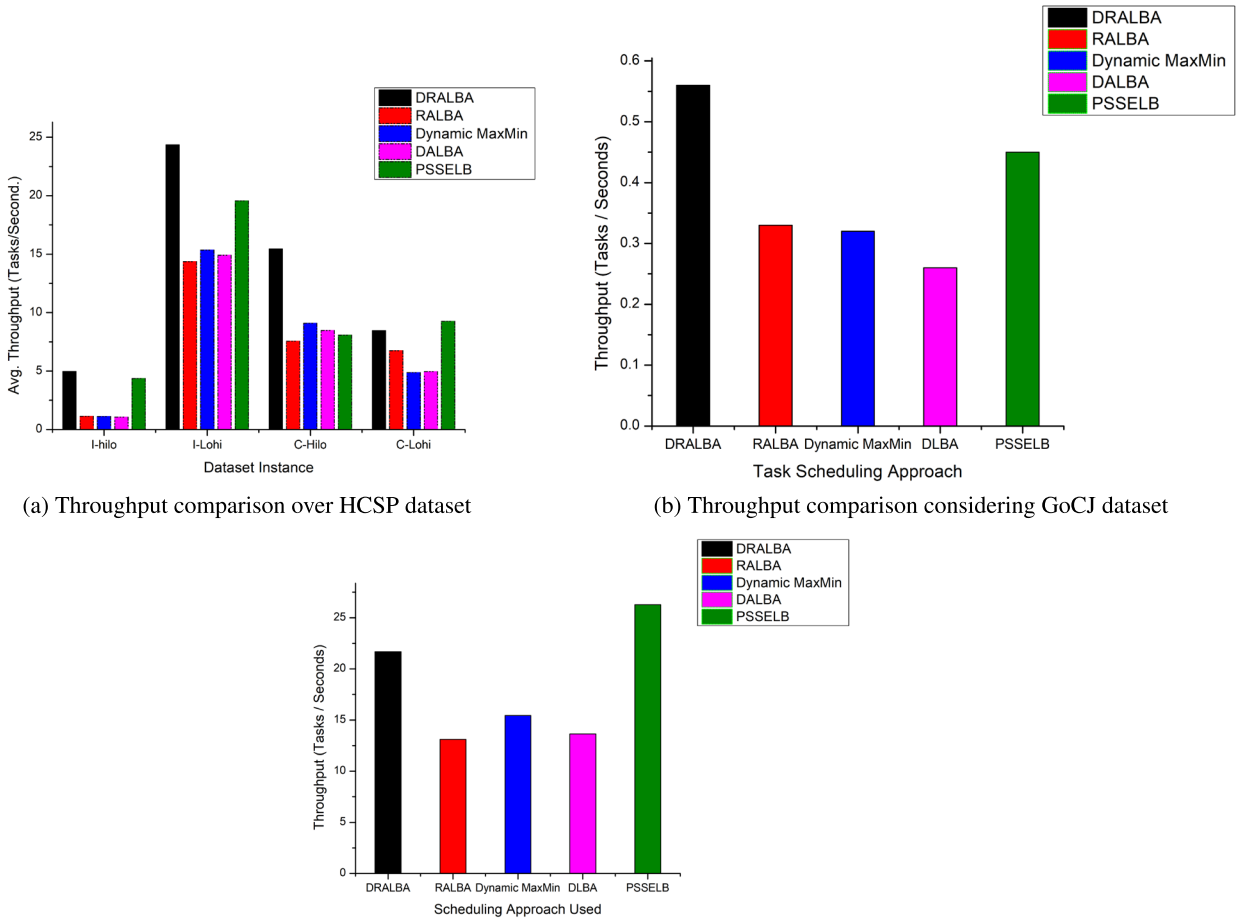
The next set of experiments are performed on Google like a realistic workload called GoCJ and are shown in Figure 3b. From the results, it can be seen that the proposed DRALBA approach has shown 57.83 %, 46.51 %, 54.93 %, and 8.43 % improvement as compared to RALBA, Dynamic MaxMin, DLBA, and PSSELB respectively. For the last set of experiments, the Synthetic workload was considered for the simulation experiments. The results concerning the Makespan are then plotted in Figure 3c. Figure 3c shows that the proposed DRALBA approach has been able to reduce the Makespan by 75.19 %, 63.84 %, and 80.28 % improvement as compared to RALBA, Dynamic MaxMin, DLBA respectively. However, the PSSELB has been able to reduce dramatically, and thus 15.85 % decrease is revealed as compared to the proposed approach.

The in-depth investigation shows that the proposed approach has been able to produce the best makespan for two of the renowned datasets. This is because *DRLABA* maps the tasks in a way that keeps all the resources busy evenly during the execution of tasks on the available resources.

All these results asserts that both these approaches have the potential to schedule the resources in an effective way and can lead to more customer satisfaction as well as efficient resources utilization.

The results concerning the Throughput obtained for all the presented approaches on three different datasets (i.e., HCSP dataset instances (i.e., C-hilo, C-lohi, I-hilo, and I-lohi) and GoCJ and on Synthetic workload) are shown in Figure 4.

Figure 4a shows that DRALBA has been able to increase the throughput on the average 48.67 %, 57.30 %, 58.34 %, and 17.69 % (for the HCSP dataset instances) as compared to the RALBA, Dynamic MaxMin, DLBA, and PSSELB respectively. For the C-Lohi dataset, the PSSELB showed 8.67 % improvement as compared to the proposed approach and also has been able to lead the rest of the two compared approaches. This is because in C-Lohi dataset, most of the jobs are of a shorter size and thus PSSELB executes shorter deadline tasks very quickly. Dynamic MaxMin and PSSELB have shown moderate performance behaviour. The RALBA approach shown a very poor Makespan performance on the



(a) Throughput comparison over HCSP dataset

(b) Throughput comparison considering GoCJ dataset

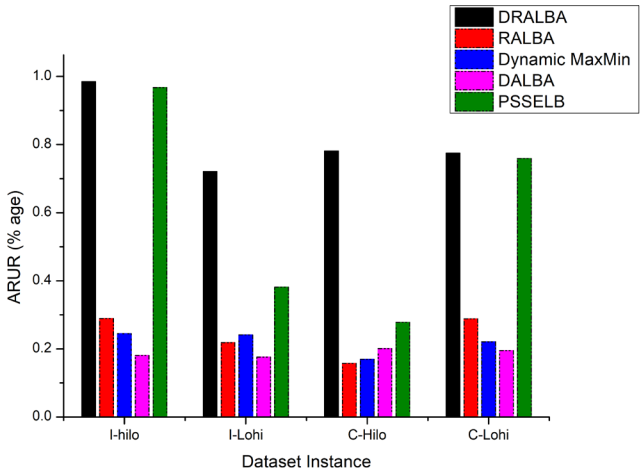
(c) Throughput comparison using Synthetic workload

FIGURE 4. **Throughput comparison on various datasets.**

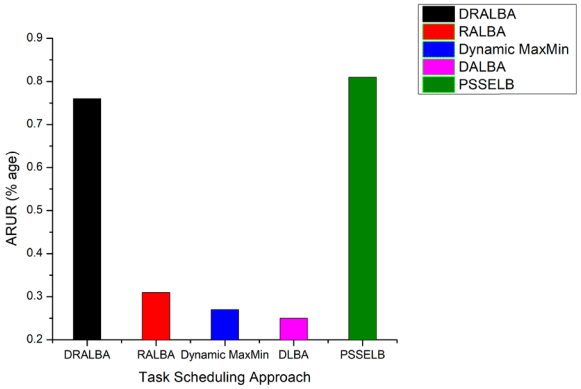
C-hilo dataset as compared to the rest of the approaches; however, behaviour similar to Dynamic MaxMin and DLBA is observed for the other three dataset instances.

The next set of experiments are performed on Google like a realistic workload called GoCJ and are shown in Figure 4b. From the results, it can be seen that the proposed DRALBA approach has shown 72.09 %, 78.10 %, 117.44 %, and 24.67 % improvement as compared to RALBA, Dynamic MaxMin, DLBA, and PSSELB respectively. The last set of simulation experiments were performed on the Synthetic workload, and the results concerning the attained Throughput are shown in Figure 4c. Figure 4c revealed that the proposed DRALBA approach has been able to increase the Throughput by 65.34 %, 40.31 %, and 58.80 % as compared to RALBA, Dynamic MaxMin, and DLBA respectively. Again similar to Makespan results for the synthetic workload, the PSSELB has been able to reduce dramatically, and thus 17.49 % decrease is revealed as compared to the proposed approach.

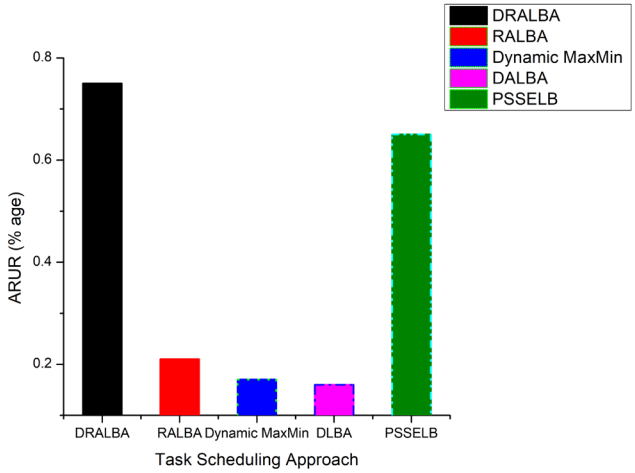
Another important parameter used for the evaluation of task scheduling heuristics is average resource utilization (ARUR). The simulation experiments were executed on four different instances of the HCSP dataset, and results are plotted on Figure 5a. The proposed DRALBA and PSSELB approach resulted in higher ARUR (0.98 and 0.96, respectively) as compared to the other contemporary approaches. Poor resource utilization (0.11) is observed for the DLBA approach. The RALBA approach has shown moderate ARUR (0.28) performance for the I-Hilo dataset and the ARUR has dropped to 0.11 for the C-Hilo dataset instance. On the average 80 %, 23 %, 19 %, 17 %, and 60 % resource utilization has been observed for the DRALBA, RALBA, Dynamic Maxmin, DLBA, and PSSELB respectively. All these facts show that the proposed DRALBA followed by the PSSELB approach involves all the computing resources equally during the task scheduling and thus very load-balanced scheduling of tasks is obtained. Moreover, the ARUR also depends on the properties of jobs in the datasets. For I-hilo and C-Lohi datasets, the proposed approach and PSSLEB obtained higher ARUR. Though the ARUR is lowered for the PSSELB on the C-Hilo and i-lohi datasets and the proposed DRALBA approach manifests steady performance for the HCSP dataset instances. From the above discussion, we can conclude that



(a) ARUR comparison over HCSP dataset



(b) ARUR comparison considering GoCJ dataset



(c) ARUR comparison using Synthetic workload

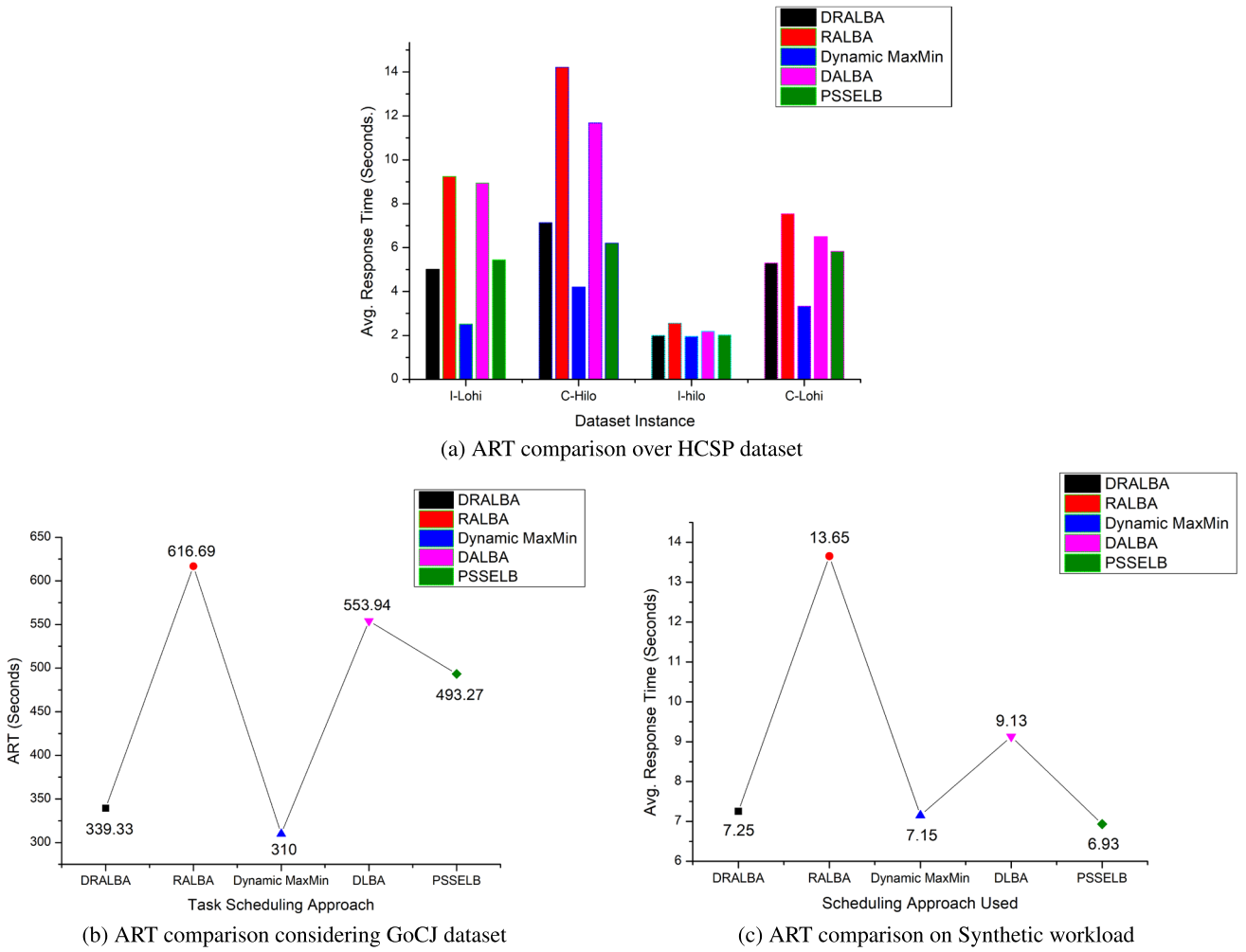
FIGURE 5. **ARUR comparison on various datasets.**

DRALBA provides a more steady and scalable performance in terms of resource utilization.

The experiments were then executed on google like real dataset, namely GoCJ, to investigate the performance of the proposed approach against the other contemporary approaches. The results concerning the attained ARUR of all the compared approaches, including the proposed approach are illustrated in Figure 5c. Again the proposed approach has dominated the rest of the approaches except the PSSELB approach. This time the PSSELB has shown an improvement of 5 % against the proposed approach. From the results, it can be seen that the proposed DRALBA and PSSELB have shown 58.83 %, 64.85 %, 67.45 %, and 61.16 %, 66.84 %, 69.29 % against improvement as compared to RALBA, Dynamic MaxMin, and DLBA respectively. The simulation experiments were then extended and executed on the synthetic workload to see a more close look at the performance of the proposed approach against the available contemporary approaches. The experiments were then performed and obtained ARUR results were then plotted in Figure 5b. Figure 5b reveals that the proposed DRALBA approach has been able to increase the ARUR by 72.00 %, 77.33 %, 78.67 %, and 13.33 % as compared to RALBA, Dynamic MaxMin, DLBA and PSSELB respectively.

The users of the Cloud are mainly interested in the response time. The final set of the simulation is executed for all the compared approaches and average response time (ART) results are plotted in Figure 6. Like other parameters, the simulations were performed on the three chosen datasets considering ART evaluation metric, and the obtained results were plotted in Figure 6a, Figure 6c, and Figure 6b.

The discussion about each of the attained results is illustrated as follows. Figure 6a plots the behaviour of the proposed DRALBA and other four contemporary approaches considering the ART results. This time the Dynamic Maxmin approach has dominated the rest of the task scheduling heuristics on all of the HCSP dataset instances. The average ART of all the datasets is 4.85, 15.89, *2.83*, 10.33, and 4.87 for



(a) ART comparison over HCSP dataset

(b) ART comparison considering GoCJ dataset

(c) ART comparison on Synthetic workload

FIGURE 6. Avg response time comparison on various datasets.

DRALBA, RALBA, Dynamic Maxmin, DLBA, and PSSELB respectively. Despite this fact, the proposed approach still has outperformed having an ART of *4.85* (second-best ART) against the rest of compared task scheduling approaches. The Dynamic Maxmin utilizes the faster machines more frequently and thus provides the best response time; however, most of the slower machines remain idle which leads to poor makespan, and also the average resource utilization is decreased. The DRALBA approach employs a load-balanced task mapping over the available resources, thus utilizing all the machines equally achieving higher resource utilization, the lowest execution time for task scheduling, and a moderate ART.

The experiments were then extended and executed using the GoCJ dataset and the ART comparison results are then plotted in Figure 6c. The results have shown an improvement of 44.97 %, 38.74 %, and 31.20 % in the average response time against RALBA, DLBA, and PSSELB respectively. However, the Dynamic Maxmin lead the proposed approach by 9.47 %.

For a more in-depth investigation, the experiments were then performed for all the contemporary approaches and compared against the proposed DRALBA approach by employing a synthetic workload dataset. The results concerning the average response time are illustrated in Figure 6b. From the results in Figure 6b, it is observed that DRALBA and Dynamic Maxmin have shown almost similar behavior. The PSSELB has revealed a slight improvement (3-4 %) as compared to DRALBA and Dynamic Maxmin approaches. The RALBA and DLBA have shown poor response time against the rest of the scheduling approaches. All these results assert that the proposed DRALBA approach has shown remarkable performance against the compared task scheduling heuristics concerning ARUR, average response time, Makespan, and Throughput.

The discussion concerning the obtained results for the compared approaches is presented to highlight important aspects of the each of the algorithm and draw the conclusions about the behavior of each of the algorithm in depth. Moreover, see how the selection of specific dataset

for the evaluation impacts the behavior of each of the approach.

Among the compared approaches, the proposed approach has shown substantial performance as compared to the rest of the approaches for all the datasets concerning the Makespan, ARUR, ART, and throughput metrics. Similarly, the PSSELB has also shown a consistent behavior for all the datasets and been a keen competitor against the proposed approach.

The Cloud service providers are mainly interested in efficient utilization of the available resources as well as their customer satisfaction. If the resources are not utilized efficiently it may to lead to high response time and large amount of energy footprints. The proposed approach being a resource aware scheduling approach has the ability to map the task in a balanced way keeping in view the available resources. This fact is endorsed by the average resource utilization results (i.e., 0.98 for HCSP dataset, 0.75 for Synthetic workload (improve ARUR by 72.00 %, 77.33 %, 78.67 %, and 13.33 % as compared to RALBA, Dynamic MaxMin, DLBA and PSSELB respectively), and 0.77 for GoCJ (i.e., the second best attained ARUR)). For large dataset like HCSP which is comprised of 1024 tasks and 32 VMs, the ARUR has raised to 98 % which confirms its scalability and the ability to map the resources in a balanced way.

From the comparative analysis it can be seen that DRALBA consumes 54.82 %, 50.16 %, 52.06 %, and 14.90 % less execution time (for the HCSP dataset instances) and 57.83 %, 46.51 %, 54.93 %, and 8.43 % for improvement (for GoCJ dataset) as compared to the RALBA, Dynamic MaxMin, DLBA, and PSSELB respectively. The PSSELB approach has shown improved results for the synthetic workload dataset being a second choice for the scheduling. This means that load-balanced mapping of the tasks on the available not only lead to better resource utilization but also decreases the execution time of the tasks.

V. CONCLUSION AND FUTURE WORK

In this research, a resource-aware dynamic task scheduling approach is proposed and implemented. Extensive simulations have been performed using three renowned large-scale dataset to see the performance of the proposed approach against the contemporary approaches. The DRALBA approach employs a load-balanced task mapping over the available resources, thus utilizing all the machines equally achieving higher resource utilization, the lowest execution time for task scheduling, and a moderate ART. The *DRALBA* approach has revealed significant improvements in terms of attained ARUR, throughput, and Makespan. For the synthetic dataset a minimum of 13.33 % against the PSSELB approach whereas significant improvement i.e., 78.67 % has been seen against the DLBA approach. The task scheduling in multi-Cloud environment is challenging and in our future work, we will propose an efficient task scheduling approach. The future work also aims to propose a task and resource-aware scheduling approach for efficient mapping of tasks on VMs in CDCs.

REFERENCES

[1] J. Lee, "A view of cloud computing," *Int. J. Netw. Distrib. Comput.*, vol. 1, no. 1, pp. 2–8, 2013.
[2] M. Ibrahim, "SIM-cumulus: A large-scale network-simulation-as-a-service," Ph.D. dissertation, Dept. Comput. Sci., Capital Univ. Sci. Technol., Islamabad, Pakistan, 2019.
[3] M. Ibrahim, M. A. Iqbal, M. Aleem, M. A. Islam, and N.-S. Vo, "MAHA: Migration-based adaptive heuristic algorithm for large-scale network simulations," *Cluster Comput.*, vol. 23, no. 2, pp. 1251–1266, Jun. 2020.
[4] M. Ibrahim, M. Imran, F. Jamil, Y. J. Lee, and D.-H. Kim, "EAMA: Efficient adaptive migration algorithm for cloud data centers (CDCs)," *Symmetry*, vol. 13, no. 4, p. 690, Apr. 2021.
[5] E. D. Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, P. Simoens, and B. Dhoedt, "Dynamic auto-scaling and scheduling of deadline constrained service workloads on IaaS clouds," *J. Syst. Softw.*, vol. 118, pp. 101–114, Aug. 2016.
[6] A. Sajjad, A. A. Khan, and M. Aleem, "Energy-aware cloud computing simulators: A state of the art survey," *Int. J. Appl. Math., Electron. Comput.*, vol. 6, no. 2, pp. 15–20, Jun. 2018.
[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.
[8] S. Kang, B. Veeravalli, and K. M. M. Aung, "Dynamic scheduling strategy with efficient node availability prediction for handling divisible loads in multi-cloud systems," *J. Parallel Distrib. Comput.*, vol. 113, pp. 1–16, Mar. 2018.
[9] A. Semmoud, M. Hakem, B. Benmammar, and J. Charr, "Load balancing in cloud computing environments based on adaptive starvation threshold," *Concurrency Comput., Pract. Exper.*, vol. 32, no. 11, p. e5652, Jun. 2020.
[10] S. Nabi and M. Ahmed, "OG-RADL: Overall performance-based resource-aware dynamic load-balancer for deadline constrained cloud tasks," *J. Supercomput.*, pp. 1–33, Jul./Aug. 2021.
[11] M. A. Alsaih, R. Latip, A. Abdullah, S. K. Subramaniam, and K. A. Alezabi, "Dynamic job scheduling strategy using jobs characteristics in cloud computing," *Symmetry*, vol. 12, no. 10, p. 1638, Oct. 2020.
[12] M. Ibrahim, S. Nabi, A. Baz, N. Naveed, and H. Alhakami, "Towards a task and resource aware task scheduling in cloud computing: An experimental comparative evaluation," *Int. J. Netw. Distrib. Comput.*, vol. 8, no. 3, p. 131, 2020.
[13] M. Ibrahim, S. Nabi, A. Baz, H. Alhakami, M. S. Raza, A. Hussain, K. Salah, and K. Djemame, "An in-depth empirical investigation of state-of-the-art scheduling approaches for cloud computing," *IEEE Access*, vol. 8, pp. 128282–128294, 2020.
[14] F. Luo and J. Yi, "A heuristic task dispatching algorithm in the clouds," in *Proc. IEEE Int. Conf. Prog. Informat. Comput.*, May 2014, pp. 498–502.
[15] K. Gardner, J. A. Jaleel, A. Wickeham, and S. Doroudi, "Scalable load balancing in the presence of heterogeneous servers," *Perform. Eval.*, vol. 145, Jan. 2021, Art. no. 102151.
[16] P. Y. Zhang and M. C. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 772–783, Apr. 2018.
[17] A. Hussain, M. Aleem, A. Khan, M. A. Iqbal, and M. A. Islam, "RALBA: A computation-aware load balancing scheduler for Cloud computing," *Cluster Comput.*, vol. 21, no. 3, pp. 1667–1680, 2018.
[18] S. K. Mishra, A. Khan, B. Sahoo, and D. Puthal, "Time efficient dynamic threshold-based load balancing technique for cloud computing," in *Proc. Int. Conf. Comput., Inf. Telecommun. Syst. (CITS)*, 2017, pp. 161–165.
[19] Y. Mao, X. Chen, and X. Li, "Max–min task scheduling algorithm for load balance in cloud computing," in *Proc. Int. Conf. Comput. Sci. Inf. Technol.*, in Advances in Intelligent Systems and Computing, vol. 255. New Dehli, India: Springer, 2014, pp. 457–465.
[20] S. Elmougy, S. Sarhan, and M. Joundy, "A novel hybrid of shortest job first and round robin with dynamic variable quantum time task scheduling technique," *J. Cloud Comput.*, vol. 6, no. 1, pp. 1–12, Dec. 2017.
[21] A. Thakur and M. S. Goraya, "A taxonomic survey on load balancing in cloud," *J. Netw. Comput. Appl.*, vol. 98, pp. 43–57, Nov. 2017.
[22] E. Y. Daraghmi and S.-M. Yuan, "A small world based overlay network for improving dynamic load-balancing," *J. Syst. Softw.*, vol. 107, pp. 187–203, Sep. 2015.

[23] N. Alaei and F. Safi-Esfahani, "RePro-active: A reactive–proactive scheduling method based on simulation in Cloud computing," *J. Supercomput.*, vol. 74, no. 2, pp. 801–829, 2018.
[24] S. Rehman, N. Javaid, S. Rasheed, K. Hassan, F. Zafar, and M. Naeem, "MinMin scheduling algorithm for efficient resource distribution using Cloud and fog in smart buildings," in *Proc. Int. Conf. Broadband Wireless Comput., Commun. Appl.* Taichung, Taiwan: Springer, 2018, pp. 15–27.
[25] H. Chen, F. Wang, N. Helian, and G. Akanmu, "User-priority guided min-min scheduling algorithm for load balancing in cloud computing," in *Proc. Nat. Conf. Parallel Comput. Technol. (PARCOMPTECH)*, Feb. 2013, pp. 1–8.
[26] O. Elzeki, M. Rashad, and M. Elsoud, "Overview of scheduling tasks in distributed computing systems," *Int. J. Soft Comput. Eng.*, vol. 2, no. 3, pp. 470–475, 2012.
[27] M. Ibrahim, S. Nabi, R. Hussain, M. S. Raza, M. Imran, S. M. A. Kazmi, A. Oracevic, and F. Hussain, "A comparative analysis of task scheduling approaches in cloud computing," in *Proc. 20th IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput. (CCGRID)*, May 2020, pp. 681–684.
[28] V. Polepally and K. S. Chatrapati, "Dragonfly optimization and constraint measure-based load balancing in Cloud computing," *Cluster Comput.*, vol. 22, no. 1, pp. 1099–1111, 2019.
[29] R. K. Naha and M. Othman, "Cost-aware service brokering and performance sentient load balancing algorithms in the cloud," *J. Netw. Comput. Appl.*, vol. 75, pp. 47–57, Nov. 2016.
[30] M. Yazdanbakhsh, R. K. M. Isfahani, and M. Ramezanpour, "MODE: A multi-objective strategy for dynamic task scheduling through elastic cloud resources," *Majlesi J. Elect. Eng.*, vol. 14, no. 2, pp. 127–141, 2020.
[31] R. K. Naha and M. Othman, "Brokering and load-balancing mechanism in the cloud–revisited," *IETE Tech. Rev.*, vol. 31, no. 4, pp. 271–276, Jul. 2014.
[32] Y. Chen and R. H. Katz, "Analysis and lessons from a publicly available Google cluster trace," EECS Dep. Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2010-9594, 2010, p. 11.
[33] A. Hussain and M. Aleem, "GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures," *Data*, vol. 3, no. 4, p. 38, 2018.
[34] C. Sang, J. Wu, J. Li, A. K. Bashir, F. Luo, and R. Kharel, "RALaaS: Resource-aware learning-as-a-service in edge-cloud collaborative smart connected communities," in *Proc. GLOBECOM-IEEE Global Commun. Conf.*, Dec. 2020, pp. 1–6.
[35] H. Liao, Z. Zhou, X. Zhao, L. Zhang, S. Mumtaz, A. Jolfaei, S. H. Ahmed, and A. K. Bashir, "Learning-based context-aware resource allocation for edge-computing-empowered industrial IoT," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4260–4277, May 2020.
[36] A. K. Bashir, R. Arul, S. Basheer, G. Raja, R. Jayaraman, and N. M. F. Qureshi, "An optimal multitier resource allocation of cloud RAN in 5G using machine learning," *Trans. Emerg. Telecommun. Technol.*, vol. 30, no. 8, p. e3627, Aug. 2019.