Received 24 January 2024, accepted 7 February 2024, date of publication 9 February 2024, date of current version 16 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3364700

RESEARCH ARTICLE

Parallel Enhanced Whale Optimization Algorithm for Independent Tasks Scheduling on Cloud Computing

ZULFIQAR ALI KHAN[1], **IZZATDIN ABDUL AZIZ**[1], **NURUL AIDA BT OSMAN**[1], **AND SAID NABI**[2]

[1]Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Malaysia
[2]Department of Computer Science and Information Technology, Virtual University of Pakistan, Lahore 44000, Pakistan

Corresponding author: Zulfiqar Ali Khan (zulfiqar_22005381@utp.edu.my)

This work was supported by the STIRF Research Grant project, Cost Centre: 015LA0-035, Universiti Teknologi PETRONAS.

ABSTRACT Cloud computing has been imperative for computing systems worldwide since its inception. The researchers strive to leverage the efficient utilization of cloud resources to execute workload quickly in addition to providing better quality of service. Among several challenges on the cloud, task scheduling is one of the fundamental NP-hard problems. Meta-heuristic algorithms are extensively employed to solve task scheduling as a discrete optimization problem and therefore several meta-heuristic algorithms have been developed. However, they have their own strengths and weaknesses. Local optima, poor convergence, high execution time, and scalability are the predominant issues among meta-heuristic algorithms. In this paper, a parallel enhanced whale optimization algorithm is proposed to schedule independent tasks in the cloud with heterogeneous resources. The proposed algorithm improves solution diversity and avoids local optima using a modified encircling maneuver and an adaptive bubble net attacking mechanism. The parallelization technique keeps the execution time low despite its internal complexity. The proposed algorithm minimizes the makespan while improving resource utilization and throughput. It demonstrates the effectiveness of the proposed PEWOA against the best performing enhanced whale optimization algorithm (WOAmM) and Multi-core Random Matrix Particle Swarm Optimization (MRMPSO). The algorithm consistently produces better results with varying number of tasks on GoCJ dataset, indicating better scalability. The experiments are conducted in CloudSim utilizing a variety of GoCJ and HCSP instances. Various statistical tests are also conducted to evaluate the significance of the results.

INDEX TERMS Task scheduling, meta-heuristic, whale optimization algorithm, cloud computing.

I. INTRODUCTION

Cloud computing has become a prime resource for a variety of applications including banking, healthcare, entertainment, and E-commerce etc. It provides numerous sorts of services on a pay-per-use basis to both users and applications by utilizing its computing, storage, and bandwidth resources [1]. The three services models are Platform as a Service, Infrastructure as a Service and Software as a Service [2], while the deployment models are private, public, community and hybrid [3]. Cloud service providers offer different levels of services with specific Quality of Service (QoS) parameters to meet the varying needs and expectations of their users. Users demand better Quality of Service (QoS), while cloud providers aim to provide scalable on demand services by employing minimum number of resources. Therefore,

The associate editor coordinating the review of this manuscript and approving it for publication was Nitin Gupta.

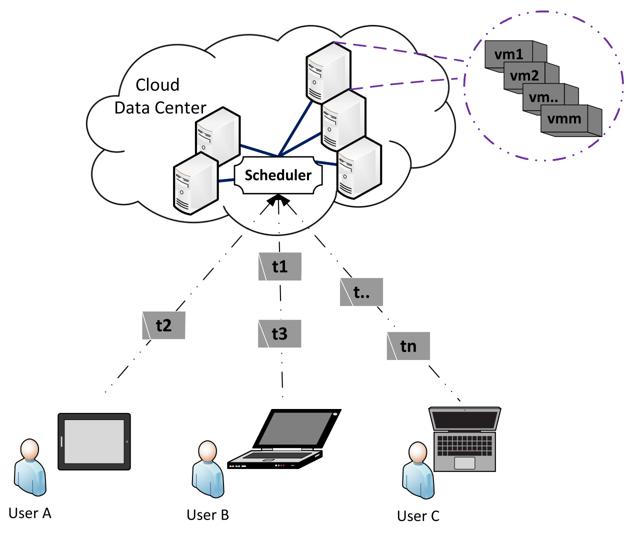

FIGURE 1. **Mapping of tasks on cloud VMs.**

a guarantee is warranted in the form of Service Level Agreement (SLA) to specify the QoS parameters for the provided service.

The philosophy of cloud computing ensures provisioning of the right number of resources as per the need at a particular time, but datacenters often overprovision the resources to avoid SLA breaches. These services can scale up and down dynamically according to user requirements. The users rent the resources for a specific period by sending a list of tasks to the cloud; a broker sends the received tasks and available virtual machines to the scheduler that maps the tasks to virtual machines. The mapping by the scheduler plays a crucial role in the overall efficiency of the datacenter. Thus, providing a consistent mapping of tasks to virtual machines with varying workload is crucial for the overall system scalability.

The role of scheduler has been a critical factor in determining the goal of executing workloads swiftly in addition to achieving optimal resource utilization [4]. Virtualization [5] of hardware resources is the core technology for cloud resource sharing where multiple virtual machines are created on a single computing node to allow running multiple tasks from multiple users as shown in Figure 1. Cloud data centers receive hundreds of thousands of tasks to run on the virtual machines on a daily basis. The huge number of tasks and large number of heterogeneous virtual machines make the task scheduling an NP-hard problem [6]. To meet user need with minimum number of resources, the resource utilization of employed resources should be increased [7]. A data obtained over six months of duration from over 5000 cloud servers revealed that the servers were utilized 10-50% of their maximum capacity [3].

Traditional static task scheduling algorithms such as Round Robin (RR), Min-Min, and Max-Min etc. provide optimal mapping of tasks to virtual machines, but these static algorithms provide poor resource utilization and cannot be used widely in the dynamic cloud environments [8]. On the other hand, dynamic task scheduling algorithms like SLA-RALBA [9], OG-RADL [10], and D-RALBA [11] are effective to deal with dynamic task scheduling on the cloud with better resource utilization, however they are deterministic and feasible only when the number of tasks and virtual machines are below a certain threshold.

Heuristic based solutions are tailored for specific problems and yield optimal results; however, for NP-hard problems, their viability is limited to below a certain threshold. On the contrary, meta-heuristic algorithms are problem independent techniques that prove promising in areas where integer programming cannot cope with the sheer number of feasible solutions in a near optimal time frame [12]. These algorithms provide an acceptable solution within a reasonable amount of time with the help of random search capability [13]. Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) are among the pioneer meta-heuristic algorithms that have been consistently used for a variety of optimization problems. On the other hand, Grey Wolf Optimization, Jaya, Firefly, and Whale Optimization Algorithm (WOA) are some of the new meta-heuristic algorithms.

The strength of any meta-heuristic algorithms resides in its abilities to effectively explore and exploit the solution space; however, these techniques face local optima and premature convergence along with high execution time. These issues can be addressed with a variety of methods to efficiently diversify the solution space and select the optimum solution in a reasonable amount of time. For a better trade-off between global and local search, the operations of meta-heuristic algorithms can be improved in addition to parallelism to decrease the overall execution time. Parallelism can be employed to either parallelize computations or parallelize the population. Meta-heuristic algorithms possess intrinsic parallelism in their operations, making them even more effective. There are three parallelization topologies as: master-slave, coarse-grained, and fine-grained. In master-slave, the master node coordinates the allocated workload to multiple slaves. The master is responsible for both communication and coordination among slaves. Coarse-grained topology on the other hand divides a program into multiple chunks with little synchronization and communication. It is also called island or ring topology. Lastly fine-grained topology distributes a program into evenly small sized tasks with high level of synchronization in addition to more communication links.

Traditional algorithms such as OG-RADL [10] and DRALBA [11] schedule tasks based on the notion of Earliest Finish Time (EFT). However, this approach hampers the resource utilization of a cloud server because faster machines are more occupied than others, which also deteriorates the completion time of all tasks, referred to as makespan. Similarly, priority driven algorithms like PBFS [14] and SG-PBFS [15] perform scheduling by favoring shorter tasks to execute first, which also degrades the resource utilization and response time for bigger tasks. To handle all tasks and resources without any preferences for shorter/larger tasks or slower/faster machines, meta-heuristic algorithms have proven promising. They can optimize resource utilization and makespan of a cloud server. These algorithms work through random operations that require a proper balance between exploration and exploitation. They provide good enough solutions in an optimal time frame. In one of our previous studies [16], the Whale Optimization Algorithm (WOA) emerged as one of the new meta-heuristic algorithms that captured researchers' attention in the domain of task scheduling on cloud and fog computing. However, the exploration capability of WOA needs further refinement. In response to this, a Parallel Enhanced Whale Optimization Algorithm (PEWOA) is proposed in this study to schedule independent tasks on heterogeneous virtual machines on the cloud. It employs a modified encircling move and an adaptive bubble net attacking mechanism to effectively perform global search and local search at the appropriate times. The algorithm conducts parallel computations for all whales, utilizing a model similar to master-slave topology with minimal communication between master and slaves. The parallelism keeps the execution time lower. The proposed algorithm shows substantial improvements in the makespan, resource utilization, and throughput against WOAMm [17], RMPSO [18], MRMPSO [18], SAEA [19], and Genetic Algorithm using MapReduce framework (GAMR) [20]. A series of experiments are conducted in CloudSim using the two workload datasets of GoCJ and HCSP.

The main contributions of this work are:

1) Shortlisting the studies from literature from 2019 to 2024 that investigate task scheduling on cloud computing in general and particularly using parallel meta-heuristic algorithms.
2) Proposing a Parallel Enhanced Whale Optimization Algorithm (PEWOA) utilizing multi-threading and improvements through a modified encircling move and an adaptive bubble net attacking mechanism.
3) Extensive simulations of the proposed PEWOA against WOAmM, RMPSO, MRMPSO, SAEA, and GAMR on two workload datasets of GoCJ and HCSP.
4) Analysis of the proposed algorithm against other algorithms in terms of minimizing the makespan, reducing response time, and increasing resource utilization and throughput in a time-efficient manner.
5) Conducting statistical tests, including assessments of standard deviation, the Friedman test, and the Wilcoxon test to illustrate the significance of the results.

The rest of the paper is laid out as follows: Section II provides the related work on task scheduling on cloud computing in general and emphasis on parallel meta-heuristic algorithms. The mathematical modelling is given in Section III. Section IV illustrates the proposed algorithm, while Section V and VI present the workload selection, and experimentation and results respectively. At the end, the conclusion is provided in Section VII.

II. RELATED WORK

The adoption of meta-heuristic algorithms for task scheduling in cloud computing has been on the rise. In large-scale heterogeneous environments, discovering the optimal solution incurs a high computation cost. Hence, pursuing a near optimal solution within a reasonable timeframe emerges as a promising approach. The following are the related studies for task scheduling on cloud computing.

An enhanced Moth Search algorithm [21] improved makespan, throughput, and load balancing during task scheduling on the cloud. The proposed algorithm was enhanced by employing differential evolution, phototaxis, and levy flight. Similarly, differential evolution was combined with Electre III for scheduling independent tasks in [22]. A nature inspired Chaotic Squirrel Search Algorithm increased the velocity and convergence precision to efficiently schedule tasks on the cloud in [23]. The task scheduling problem was formulated as a multi-objective optimization problem. The early eco-system was developed through messy optimization to reduce expenses, prevent SLA violations, and minimize resource consumption. The proposed algorithm not only minimized makespan and energy consumption but also improved resource utilization, load balancing, and met deadlines.

A balanced distribution of resources is essential for minimum makespan and maximum resource utilization. OG-RADL, an Overall Performance-based Resource aware Load-balancer was proposed to schedule independent tasks in the cloud [10]. It successfully minimized the makespan and maximized the resource utilization and throughput in addition to better load balancing. However, the algorithm toke decisions based on Earliest Finish Time (EFT) for all compute-intensive tasks, that made faster machines more occupied than others. Similarly, Dynamic and Resource Aware Load Balanced Approach (DRALBA) scheduled independent tasks using a deterministic routine to optimize average resource utilization, throughput, and makespan using GoCJ and HCSP workloads. However, it also occupies the faster machines more than others with the increase in workload [11].

The authors in [24] used the Dragonfly algorithm, Biogeography-based algorithm, and Mexican Hat Wavelet to reduce both execution time and response time during task scheduling. These three techniques successfully prevented premature convergence of the solution space and minimizing the SLA violations. The combination of the Biogeography-based algorithm and Mexican Hat Wavelet Transform introduced a mutation operation to assist the Dragonfly algorithm in avoiding local optima. However, in the given scenario, the mutation operation of a traditional genetic algorithm might be more beneficial. In [25], an adaptive regressive Holt-Winters algorithm is utilized to predict bursty or normal workload. Subsequently, the Firefly algorithm with lottery approach was applied to optimize the scheduling process, enhancing resource utilization, load balancing, and minimizing the energy consumption. However, the study did not highlight the nature of tasks. Another study based on modified Henry gas solubility optimization, improved makespan and execution cost during task scheduling [26]. Yet, the impact of the improved makespan on resource utilization and throughput was not discussed. In [27], the authors proposed two scheduling algorithms for independent deadline sensitive tasks. The first algorithm employs a greedy approach based on a linear weight sum. The second algorithm used Ant colony optimization, positive feedback mechanism, and heuristic search. The proposed algorithms minimize energy consumption and makespan.

In [18], independent task scheduling was formulated with budget constraints and addressed using two parallelized PSO algorithms. The PSO was initially enhanced using a random integer matrix (RMPSO), followed by proposing two parallel variants of RMPSO based on a Multi-core system (MRMPSO) with shared memory and a many core-GPU system (GRMPSO). The GRMPSO outperformed the MRMPSO in decreasing the total cost and running time of the algorithm. The proposed G-RMPSO used fine-grained GPU threads to accelerate RMPSO particles' computations. During experiments, the number of threads varied from 2 to 12 for OpenMP and from 4 to 20 for CUDA.

In another study, a parallel Squirrel Search Algorithm (SAEA) combined with fuzzy logic optimally scheduled independent tasks on the cloud to minimize makespan, degree of imbalance, security threats, and energy cost under high load conditions [19]. The population was divided into subgroups to evolve independently. After a specific number of iterations, the best squirrels were placed in the next sub-population by replacing the worst squirrels. Fuzzy logic was used to calculate the fitness of each squirrel based on total execution time, makespan, energy cost, degree of imbalance, and security value. However, the communication strategy in SAEA was fixed. The population of squirrels was divided into ten sub-populations to facilitate the convergence of separate groups of squirrels. Subsequently, the best squirrels were migrated to the next sub-populations to increase the exploration of search space and avoid the local optima.

Task prioritization poses a bottleneck in exploration-based scheduling approaches that use various techniques for prioritizing tasks, resulting in increased execution times. Prioritizing tasks based on the shortest execution time deemed appropriate. To address this challenge, a parallel Genetic Algorithm using MapReduce (GAMR) was proposed for cloud workflow scheduling, incorporating different priority queues to reduce the makespan [20]. In the first phase, the GA and earliest finish time approach assigned tasks to processors followed by using GA with MapReduce to assign jobs to processors in a heterogenous cloud environment. GAMR outperformed PSO, WOA, Moth-Flame Optimization (MFO), and Intelligent Water Drops (IWD). Nonetheless, only the mutation operation was parallelized in the proposed algorithm.

The unpredictable nature of workload on cloud servers is a major pitfall for reduced resource utilization and efficiency. A task scheduling strategy based on binary JAYA was implemented in [28] to alleviate the above issues. It not only increased the resource utilization, but also reduced the energy consumption and minimized the makespan. In the first stage, tasks were evenly distributed on virtual machines, subsequently executing the proposed JAYA algorithm for the best possible matchmaking among tasks and virtual machines. Both independent and dependent tasks were simulated in experiments to reduce both the makespan and energy consumption, improve load balancing, and maximize the resource utilization. However, the proposed algorithm was evaluated against the old versions of Genetic Algorithm, Particle Swarm Optimization, and Round Robin.

A hybrid firebug and Tunicate Optimization (HFTO) algorithm optimized makespan, response time, and fault tolerance [29]. The proposed algorithm offered an enhanced searching capability with faster convergence. HFTO is a preemptive technique that assigns smaller tasks to virtual machines with peak load, while assigning bigger tasks on machines with lower CPU utilization. It improved makespan, average execution time, and load balancing among the machines. The task preemption also improved both the execution time and response time.

Another scheduling technique called the Johnson Sequencing algorithm was originally used in a manufacturing unit. In [30], the Johnson Sequencing algorithm was adapted using a three step approach for task scheduling in cloud computing across three servers minimizing the completion time of all tasks. First, a precedence constraint graph was developed for identification of dependencies among jobs. Second, the jobs were assigned to servers followed by employing the Johonson Sequencing to determine the best ordering of the jobs on each server. The proposed Johonson Sequencing algorithm minimized makespan and improved resource utilization in addition to exhibiting better scalability. However, the scalability analysis was based on a limited number of jobs during simulations. In [31], a hybrid algorithm based on Genetic Algorithm and Gravitational Emulation Local Search (GELS) was developed, minimizing makespan and increasing resource utilization while scheduling task in the cloud. However, the comparative analysis only included the primitive versions of GA and PSO.

Task schedulers based on priority rules struggle to meet user satisfaction. To tackle this problem, a Priority Based Fair Scheduling (PBFS) was presented in to minimize the makespan, flow time and total tardiness [14]. However, only two dataset instances of GoCJ were utilized out of nineteen during simulations. In continuation of this study, the Priority Based Fair Scheduling (PBFS) algorithm was improved by proposing Shortest Gap-PBFS (SG-PBFS), a backfilling technique utilizing gaps in the job schedule [15]. The proposed algorithm outperformed other Shortest Gap based algorithms such as SG-SJF, SG-LJF, and SG-(Max-Min) etc. in terms of minimizing makespan, missed deadlines, reducing both delays and flowtime. However, the nature (homogeneous or heterogeneous) of virtual machines was not stated. Moreover, SG-PBFS favors shorter jobs to execute on a priority basis that may result into lower resource utilization. The experiments have not used all the instances of GoCJ workload.

In the literature, there are abundant studies based on parallel meta-heuristic algorithms for task scheduling, but the keyword "Parallel" refers to parallelism in two different perspectives. One involves parallelism among tasks' execution, while the other entails parallel execution of the scheduling algorithm. Most of the studies primarily focus on the first interpretation, which involves running multiple tasks in parallel after the scheduling decision has been made. However, this work focuses on the parallel execution of a meta-heuristic algorithm for independent task scheduling on the cloud. Therefore, [18], [19], and [20] represent the most relevant studies found in the literature that are considered for comparative analysis with the proposed Parallel Whale Optimization Algorithm. Furthermore, [18] and [19] dealt with independent tasks scheduling, while [20] addressed the scheduling of dependent tasks. Similarly, [18] and [20] parallelized the computations performed by the agents, while [19] parallelized the sub-populations of agents. The proposed PEWOA also parallelizes the computations performed on all whales in the population. Table 1 presents the summary of the related studies.

III. MATHEMATICAL MODELING

A cloud data center contains hundreds of host machines that provide various types of resources to end users. Each host machine often resides thousands of dynamically generated virtual machines [32]. Similarly, multiple hosts can collectively generate a single virtual machine [33]. Cloud providers offer different types of virtual machines with various performance and pricing specifications. This paper presents the allocation of virtual machines to incoming independent tasks. It is assumed that each task will run on a single virtual machine and cannot be divided. Task scheduling with heterogeneous resources is a combinatorial optimization problem, where the tasks and virtual machines can be expressed as Eq. (1) and Eq. (2) respectively.

$$T = \{t_1, t_2, , t_n\} \tag{1}$$

$$VM = \{vm_1, vm_2, , vm_m\} \tag{2}$$

The set T contains the number of instructions for each task, while VM represents a set of virtual machines with compute capacities in Millions of Instruction Per Second (MIPS). Generally, the number of tasks is greater than the number of vms. The sets T and VM serve are inputs for a scheduling algorithm, and an optimized mapping of all tasks over a set of vms present a final solution expressed in the form of a map, as shown in Eq. (3):

$$Map = \{(t_1, vm_2), (t_2, vm_1), , (t_n, vm_m)\} \tag{3}$$

In the solution map, the first item (task) of every tuple will be unique, while the second item (vm) can be repeated. Each task will be allocated only one vm, whereas a vm can have multiple tasks mapped to it. The Execution Time (ET) of $task_i$ on vm_j can be computed using Eq. (4).

$$ET_{task_i vm_j} = No.\ of\ Instructions\ in\ t_i/vm_j\ MIPS \tag{4}$$

It is assumed that each virtual machine will execute multiple tasks in a specific order without preemption. The Completion Time (CT) of all assigned tasks on a specific vm is expressed as Eq. (5):

$$CT_{vm_j} = \sum_{i=1}^{n}(No.\ of\ Instructions\ in\ t_i/vm_j\ MIPS) \tag{5}$$

Here faster machines will have shorter completion times as compared to slower machines. In a meta-heuristic algorithm, the agents are manipulated in various ways before finding their fitness, therefore the assigned tasks on a vm will keep on changing during the execution of an algorithm. If a replacement of task x on a vm with task y is required, the completion time of all tasks on that vm will be updated through Eq. (6):

$$\begin{aligned} CT_{vm_j} = CT_{vm_j} &- (No.\ of\ Instructions\ in\ t_x/vm_j\ MIPS) \\ &+ (No.\ of\ Instructions\ in\ t_y/vm_j\ MIPS) \end{aligned} \tag{6}$$

One of the important factors during tasks scheduling is makespan, which is the completion time of all tasks on a given set of virtual machines. It is represented by Eq. (7) [10]:

$$\text{makespan} = \max(CT_{vm_j}) \qquad \forall j \in 1, 2, \ldots, k \tag{7}$$

The unit used to represent execution time, completion time, and makespan in this paper is seconds. As it is beneficial to use a resource in its entirety before employing another instance on cloud, a higher resource utilization is favorable during task scheduling. The Average Resource Utilization (ARU) of a host machine is computed using Eq. (8) [10]:

$$\text{ARU} = (\sum_{i=1}^{m} CT_{vm_j}/m)/makespan \tag{8}$$

The sum of completion time for all vms is divided by the number of vms (m). The resulting value is then divided by makespan.

The efficiency of a system can be expressed in terms of throughput, which is the number of tasks executed per unit time. It can be expressed by Eq. (9) [11] as:

$$\text{Throughput} = Total\ No.\ of\ tasks/Makespan \tag{9}$$

Throughput is equal to the total number of tasks divided by makespan. The unit for throughput will be the number of tasks completed in one second.

After the scheduling decision, the time a vm takes to start executing a task is referred to as Response Time (RT). Multiple vms often share the same physical host and multiple

TABLE 1. Summary of related studies.

Reference	Year	Types of Task		Category of Solution		Algorithm	Highlights	Limitations	Benchmark Methods	Parameters considered
		Standalone	Workflow	Heuristic	Meta-Heuristic					
[21]	2019	✓	×	×	✓	Moth Search Algorithm + DE	Using phototaxis and Levy flight	The nature of tasks is not mentioned	PSO, WOA, RR, SJF	Makespan, throughput
[22]	2019	✓	×	×	✓	EATSD	Differential evolution is combined with Electre III	Only one benchmark technique is a valid contender in comparative analysis	Earliest Deadline First (EDF), FCFS, a hybrid algorithm based on priority dynamic queues, Simulated Annealing, PSO	Energy consumption, resource utilization, makespan
[23]	2020	×	✓	×	✓	CSSA	Multi-objective optimization with early eco-system development through messy optimization	The datasets used in the simulation are not elaborated	Hybrid GA-PSO, PSO-BAT	Energy consumption, resource utilization, makespan
[10]	2021	✓	×	✓	×	OG-RADL	An improved variant of EFT scheduled compute intensive tasks.	Faster machines tend to be occupied more than others.	DLBA, DCDLBA, Dynamic MaxMin, RALBA, PSSELB, MODE	Load balancing, meeting deadlines, response time.
[11]	2021	✓	×	✓	×	DRALBA	An improved variant of EFT scheduled compute intensive tasks.	Virtual machines with higher MIPS remain busy as compared to lower MIPS machines.	RALBA, Dynamic MaxMin, DLBA, PSSELB	Average Resource Utilization (ARU), throughput, makespan
[24]	2021	✓	×	×	✓	BMDDSF	Combination of Dragonfly algorithm, Biogeography-based algorithm, and Mexican Hat Wavelet to avoid premature convergence	The mutation phase can be improved by a generic GA	Dragonfly algorithm (DA), PSO, BAT, Memory-based Hybrid Dragonfly algorithm (MHDA), WOA, Raven Roosting Optimization (RRO), Adaptive Dragonfly Algorithm (Adaptive_DA), Chaotic Dragonfly Algorithm version 9 (CDA9)	Execution time, response time, SLA violation
[25]	2021	-	-	×	✓	NMTFOLS	Using workload predictor and Firefly algorithm with Lottery approach	The proposed meta-heuristic algorithm is not compared with any other meta-heuristic	RALBA , Probability based Load Balancing (PLB), Dynamic Two-stage Strategy	SLA violation, resource utilization, energy consumption
[26]	2021	✓	×	×	✓	Modified HGSO	Modified Henry Gas Solubility Optimization (HGSO) improves performance	All the compared techniques were the primitive versions	WOA, HGSO, MFO, FA, PSO, Salp Swarm Algorithm (SSA)	Makespan, cost
[27]	2021	✓	×	✓	✓	Based on Linear Weighted Sum and ACO	A greedy approach using linear weighted sum and positive feedback mechanism	The benchmark techniques are not state of the art	Energy-Aware scheduling algorithm for Real-time, aperiodic, and independent tasks (EARH), Energy-aware scheduling based on Learning Automata (LAEAS) for real-time independent tasks, Adaptive Task Allocation Algorithm (ATAA)	Makespan, energy consumption
[18]	2021	✓	×	×	✓	M-RMPSO, G-RMPSO	Parallelism based on multi-core system and many-core GPU system	The master node can become a bottleneck especially when the number of tasks is very small or the slaves are large in number	Fuzzy Modified PSO (FMPSO), HYBRID (MPSO+Modfied Cat Swarm Optimization)	Total cost
[19]	2021	✓	×	×	✓	PSSA + Fuzzy Logic	Use of Fuzzy logic and subgrouping of squirrels evolving independently	Fixed communication strategy among agents	SPSO, SGA, MPSO, FUGE, KARP, genetic-Michigan, ECM, MOWS, SAEDF, MGA, FMPSO algorithms	Makespan, load balancng, security, energy consumption cost
[20]	2022	×	✓	×	✓	GA + HEFT	Parallelism utlizing priority queues and MapReduce framework	The parallelization procedure is not elaborated in depth	PSO, WOA, MFO, Intelligent Water Drops	Makespan
[28]	2022	✓	×	×	✓	Modified Jaya	Binary Jaya algorithm working in the second stage after even distribution of tasks on virtual machines	Comparison with old versions of RR, Binary PSO, GA	RR, Binary PSO, GA	Energy consumption, resource utilization, makespan
[29]	2022	×	✓	×	✓	Firebug Swarm Optimization (FSO) plus Tunicate Swarm Optimization (TWO) called HFTO	Light weight and computation intensive tasks' allocation to VMs with high and low CPU utilization figures respectively	No details provided about the dataset	Hybrid TabuâĂŞHarmony task scheduling (HTHTS), Service level agreement-based Load Balancing (SLA-LB), EnergyâĂŘefficient taskâĂŘscheduling (EETS), Improved Firework Algorithm (IFA)	Makespan, load balancing, fault tolerance, response time, response time

TABLE 1. ***(Continued.)*** **Summary of related studies.**

[30]	2023	×	✓	✓	×	Dynamic Heuristic Johnson Sequencing	Adaptation of Johnson Sequencing algorithm for cloud task scheduling	The scalability analysis was based on a limited number of jobs during simulations	FCFS with 2 servers, FCFS with 3 servers, Priority Scheduling, Round Robin	Makespan, resource utilization, scalability
[31]	2023	×	✓	×	✓	GA+GELS	Hybridization of Genetic Algorithm and Gravitational Emulation Local Search (GELS)	The comparative analysis only included the primitive versions of GA and PSO	GA, PSO	Makespan, resource utilization
[14]	2023	✓	×	✓	×	PBFS	Task scheduler employing priority rules	During simulations, using only two dataset instances of GoCJ out of nineteen	SJF, LJF, Max-Min, FCFS, EDF	Makespan, turnaround time, delay
[15]	2024	✓	×	✓	×	SG-PBFS	Priority-based scheduling with a backfilling technique to utilize gaps in a job schedule	SG-PBFS favors shorter jobs to execute on a priority basis that may result into lower resource utilization. The experiments have not used all instances of GoCJ workload	SJF, LJF, Max-Min, FCFS, EDF	Makespan, turnaround time, delay, meeting deadlines

tasks can run on a single vm. Eq. (10) [10] represents the average response time of all tasks on a set of vms as follows:

$$RT = (\sum_{j=1}^{m}\sum_{i=1}^{n} RT_i)/m \tag{10}$$

The sum of the execution start times of all tasks is divided by the total number of tasks to yield average response time of a single vm. Then the sum of the average response times for each vm is divided by the total number of vms.

Table 2 lists down the description of all notations used in equations and pseudocode of the proposed algorithm.

IV. PROPOSED PARALLEL ENHANCED WHALE OPTIMIZATION ALGORITHM

The proposed algorithm is an enhanced version of Whale Optimization Algorithm (WOA) [34]. The WOA was formulated as a population based meta-heuristic algorithm inspired from humpback whales. In this approach, several whales serve as agents, each representing a prospective solution to the optimization problem. A group of whales is aware of prey's location and employes a hunting strategy called bubble net feeding. It involves two types of maneuvers: a circular movement and a shrinking move that reduce the circumference of the circle, as illustrated in Figure 2. The whales also release air bubbles that ascend from the whales to the top of the seawater. Several whales start these maneuvers, gradually ascending to sea's surface while simultaneously shrinking the circle and bubbling, effectively trapping the prey (a school of fish or krill) in a confined area.

The shrinking encircling mechanism facilitates exploitation of the solution space that is governed by a variable A using Eq. (11) and (12).

$$a = 2 - itr * (2/maxItr) \tag{11}$$

$$A = (2.0 * a * r1) - a \tag{12}$$

r1 is a random number whereas "a" linearly decreases from 2 to 0 that shrinks the circle around the prey. Similarly, A is a random value in the range of [-a,a]. If the value of A is greater than/equal to 1, a new random whale position (nRWPos) in the search space is selected using Eq. (13) to

TABLE 2. **Notation and descriptions.**

Notation	Description
T	Set of tasks
t	Task
VM	Set of virtual machines
vm	Virtual machine
$Map(t_i, \text{vm}_j)$	HashMap having task i as key and virtual machine j as value
$ET_{\text{task}_i,\text{vm}_j}$	Execution time of task i on virtual machine j
CT_{vm_j}	Completion time of all tasks allocated to virtual machine j
vm_j MIPS	The compute capacity of virtual machine j in MIPS
$Cloudlet$	Another name for task in CloudSim
cld	An instance of Cloudlet
$cldLength$	The number of instructions in a task
$wMap$	HashMap of all whales
$vmMap$	HashMap of all virtual machines
$bWsMap$	HashMap of all best whales
$gBWMap$	HashMap of all tasks and virtual machines in the global best whale
$nWPos$	New whale position
$bWPos$	Best whale position
$nRWPos$	New random whale position
$randPos$	Random position
$gBWMapPos$	Position of global best whale
$cWPos$	Current whale position
$bWMk$	Best whale makespan
$gBWValue$	Global best whale fitness
$vmMips$	Virtual machine compute capacity in MIPS
$randVm$	Random virtual machine
$r1, r2$	Random variables 1 and 2 in the range of 0 (inclusive) and 1 (exclusive)
$maxItr$	Maximum number of iterations
$\text{randNo}(0, 1)$	Random number between 0 (inclusive) and 1 (exclusive)
$vmListSize$	The size of a list of virtual machines
oLd	Old load
nLd	New load
$\text{getIndexFrMap}(\text{Map})$	Function to return the index of a key in a HashMap

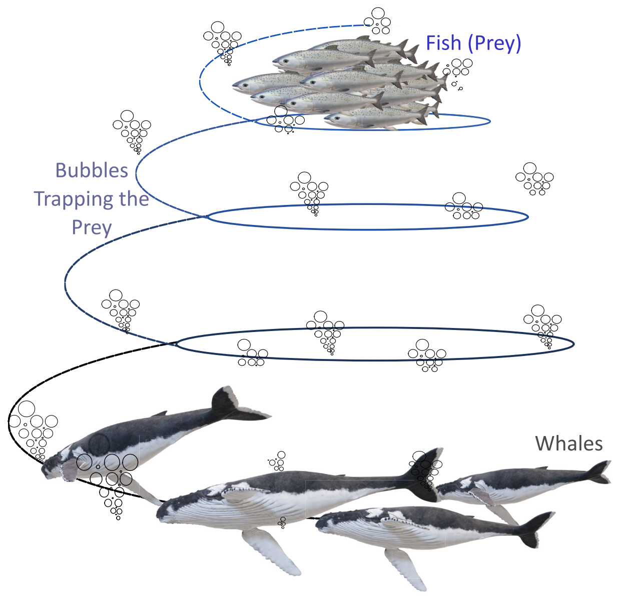

FIGURE 2. **Movement of whales during bubble net attack.**

explore the solution space.

$$nRWPos = randPos - A * |C * randPos - cWPos| \quad (13)$$

Conversely, if the value of A is less than 1, a new position will be computed using the best whale's position by Eq. (14).

$$nWPos = bWPos - A * |(C * bWPos - cWPos)| \quad (14)$$

There is another random variable p with a range of [0,1] that represents the probability of using bubble net feeding. If p is equal to/greater than 0.5 (50% probability), bubble net attack is triggered to update the whale's position (nWPos) according to Eq. (15), otherwise the whale keeps on shrinking the circle according to Eq. (14) as shown in Figure 2.

$$nWPos = |bWPos - cWPos| * e^{bl} * \cos(2\pi l) + bWPos \quad (15)$$

l is also a random number in the range [−1,1], while b is a constant defining the shape of spiral. In our proposed algorithm the value of b ranges from 1 to 2.5 to determine the logarithmic spiral shape. If the new whale position (nWPos) is outside the solution space, it is assigned a random position.

After computing the new position of a whale, its fitness is calculated. The process of updating the whales' positioning continues until the maximum number of iterations are completed. In every iteration, the best whale is selected and kept in memory.

Due to the limited exploration abilities of whale optimization algorithm, a modified encircling maneuver and an adaptive bubble net attacking mechanism are proposed in addition to parallelism. If the solution space contains local optima, WOA tends to trap in it. Unlike continuous optimization problems, combinatorial optimization faces a narrow search space with a high probability of local optima.

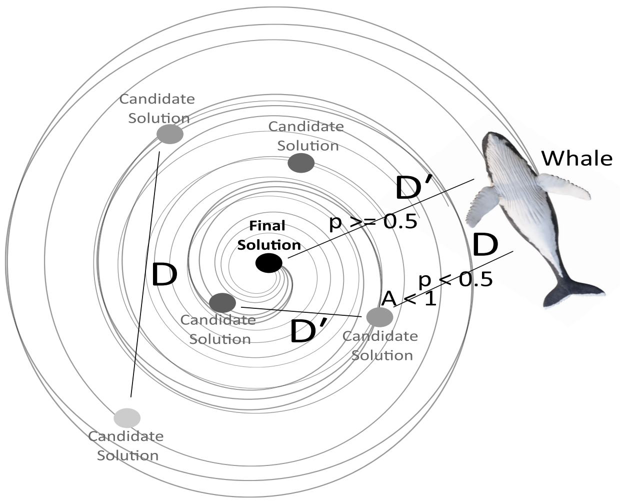

FIGURE 3. **WOA convergence behavior.**

The proposed enhancements enable PEWOA to move out of the local optimum regions. The shrinking encircling mechanism is enhanced using Eq. (16) to replace Eq. (11).

$$a = 3 - itr * (2/maxItr) \quad (16)$$

The proposed change in Eq. (16) increases the solution diversity by generating the values of a in the range of [1,3]. It improves the exploration potential of PEWOA. Further, Eq. (17) is used to modify the coefficient of the spiral updating mechanism, altering the shape of the logarithmic spiral during bubble net attacking mode.

$$b = 1 + (w100) + (w\%2.25) * (itrmaxItr) \quad (17)$$

Eq. (17) allows different shapes of the logarithmic spiral for whales, thus enabling a better balance between exploration and local search. A lower value of b favors global search, while a higher value exploits the best known solutions in the search space. Eq. (17) keeps the value of a below 1 for roughly 60% of the time, while it reaches up to 3 in the later stages to refine the existing solutions.

In the proposed PEWOA, a separate thread using Java Executor Framework [35] is allocated to the encircling and bubble net attacking maneuvers of each whale using a distinct set of values for faster convergence. It utilizes a master-slave parallelization model with multiple threads and a shared memory to explore different regions of the solution space. It provides minimal communication among threads and between master and slave nodes as depicted in Figure 4.

Unlike the master-slave model used in MRMPSO [18], the threads in PEWOA do not send back the result to a master node; instead, every thread updates a shared memory to store the global best solution. If any thread is stuck in local optima, others could still perform the exploration of the solution space and converge to the global optimal solution. The shared variables and maps among the whales are accessed via locks to ensure data consistency. Whale

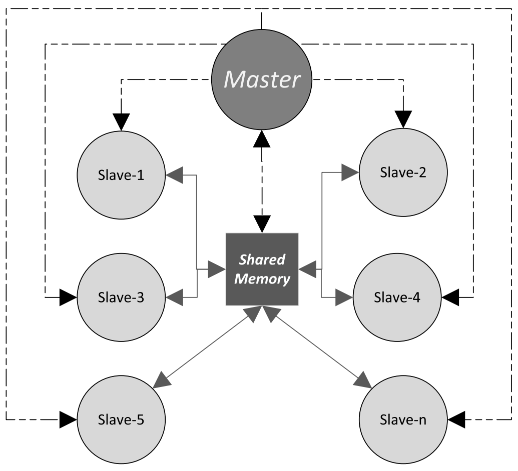

FIGURE 4. Parallelization model.

Optimization Algorithm, in general, involves a number of complex operations, making it a computation-intensive procedure. Therefore, a parallelization strategy is valuable to reduce the execution time of PEWOA. Following 1 is the pseudocode of the proposed algorithm.

A sets of tasks (T) and virtual machines (VM) serve as the inputs for PEWOA, while the final schedule is represented as a hashmap with T as keys and VM as values. In lines (1)-(4), hashmaps are declared for whales (wMap), virtual machines (vmMap), best whales (bWsMap), and global best whale (gBWMap). The number of whales and maximum iterations are specified in line 5. Best whale makespan (bWMk) and global best value (gBValue) are initialized with maximum values in line 6. In line 7, all maps are initialized.

A while loop spanning over line 8 to 52 iterates 220 times. At line 9, a pool of threads is created based on the population size of whales. A for loop for each whale begins at line 10 and continues until line 52, implementing various types of changes for each whale. The shrinking encircling parameter is updated in each iteration ranging from 3 to 1 (line 11) followed by the declaration of three random numbers r1, r2, and p (probability) in the range of [0,1] (line 12). At lines 13-14, A and C are declared as coefficients with ranges $[-3,3]$ and [0,2] respectively, while l is variable with values $[-1,1]$ (line 15). b ranges from 0 to 3 (line 16) and variables Drand, D, D′, nRWPos, nWPos, and randVm are initialized to zero (line 17). The description of these variables is given in Table 1.

A for loop iterates (line 18) over all tasks in a whale to manipulate the assignment of virtual machines for each task. At line 19 and 20, the current task and vm are assigned to cloudlet and vm variables respectively. Line 21 to 32 presents an if construct based on the value of p. If p is less than 0.5 and the absolute value of A in a nested if statement (line 22) is greater than or equal to 1, the distance is computed from the current whale position to any random whale position (line 23). Based on the random distance, the new whale

Algorithm 1 Parallel Enhanced Whale Optimization Algorithm

```
Input: Set of Tasks (T), set of virtual machines (VM)
Output: Map(T, VM)
 1: $wMap \leftarrow$ (Integer, Map(Cloudlet, Vm)) = null
 2: $vmMap \leftarrow$ (Integer, Map(Integer, Double)) = null
 3: $bWsMap \leftarrow$ (Integer, Double) = null
 4: $gBWMap \leftarrow$ (Integer, Map(Cloudlet, Vm)) = null
 5: whales = 60, maxItr = 220
 6: $bWMk, gBValue \leftarrow$ max_value
 7: Initialize $vmMap$, $bWsMap$, $gBWMap$, and $wMap$
 8: while itr < maxItr do
 9:     Pool of threads (Whales)
10:     for $w = 0 to noOfthreads$ do
11:         $a \leftarrow 3.0 - (\text{itr} \cdot \frac{2.0}{\text{maxItr}})$
12:         $r1, r2, p \leftarrow \text{randNo}(0, 1)$
13:         $A \leftarrow (2.0 \cdot a \cdot r1) - a$
14:         $C \leftarrow 2.0 \cdot r2$
15:         $l \leftarrow (\text{randNo}(0, 1) \cdot 2.0) - 1.0$
16:         $b \leftarrow 1 + \frac{w}{100} + \frac{w \mod 2.25 \cdot \text{itr}}{\text{maxItr}}$
17:         $Drand, D, D', nRWPos, nWPos, \text{randVm} \leftarrow 0$
18:         for each task do
19:             $cld \leftarrow$ currentCloudlet
20:             $vm \leftarrow$ CloudletVm
21:             if $p < 0.5$ then
22:                 if $|A| \geq 1$ then
23:                     $Drand \leftarrow |C \cdot \text{randVm} - cWPos|$
24:                     $nRWPos \leftarrow \text{randVm} - A \cdot Drand$
25:                 else if $|A| < 1$ then
26:                     $D \leftarrow |C \cdot \text{gBWMapPos} - cWPos|$
27:                     $nWPos \leftarrow \text{gBWMapPos} - A \cdot D$
28:                 end if
29:             else if $p \geq 0.5$ then
30:                 $D' \leftarrow |\text{gBWMapPos} - cWPos|$
31:                 $nWPos \leftarrow D' \cdot ebl \cdot \cos(2\pi \cdot l) +$ gBWMapPos
32:             end if
33:             if $nWPos >$ vmListSize$or nWPos < 0$ then
34:                 $nWPos \leftarrow$ randPos
35:             end if
36:             randVm $\leftarrow$ vmList($nWPos$)
37:             $oLd \leftarrow \frac{\text{cldLength}}{\text{vmMips}}$
38:             $nLd \leftarrow \frac{\text{cldLength}}{\text{randVmMips}}$
39:             update_vmMap(vm, oLd, nLd)
40:             update $wMap$
41:         end for
42:         synchronized (lock)
43:         $bWMk \leftarrow$ getPBMap($vmMap$)
44:         if $bWMk < bWsMap$ then
45:             Update $bWsMap$
46:             if $bWMk < gBWValue$ then
47:                 $gBValue \leftarrow bWMk$
48:                 Update $gBWMap$
49:             end if
50:         end if
51:     end for
52: end while
```

position is calculated at line 24 using A. If p is less than 0.5 and the absolute value of A is less than 1 in the else part of the second if clause (line 25), the distance is computed from the current whale to the best whale found so far (line 26). At line 27 the new whale position is computed by utilizing the best whale positioning and A. The nested if clause ends at line 28. If p is greater than or equal to 0.5 in the else part of the first if clause, the algorithm enters the bubble net attacking mode (line 29). Now the distance of the current whale is calculated from the best whale without using the value of C (line 30). The new whale position is computed using the location of the best whale, the corresponding distance (D′), and a spiral manoeuvre ($e^{bl} * \cos(2 * \pi * l)$) at line 31. The if statement at line (21) ends at line 32.

Another if statement (line 33-35) checks the new whale position. If it is outside the solution space, a random position is assigned to the whale. Based on the new whale position, the relevant vm is selected from the list of virtual machines (line 36). The execution time of a task on the already assigned vm is calculated and saved in oLd (line 37). At line 38, the execution time of the task on the new Vm is computed and stored in nLd. A function update_vmMap() propagates the new load of task by adding and removing the execution times on the two virtual machines (line 39). The whales map is updated at line 40 and the loop ends at line 41. The code from line 42 to 51 can only be executed by a single thread at a time. At line 43, the best whale's makespan is calculated using *getPBMap*() function. If the new makespan is less than the whale's old makespan (line 44), then the best whales map will be updated with the new fitness value (line 45). At line 46, if the newly calculated makespan is less than the global best makespan, it will also be assigned the new value of makespan (line 47). The new global best whale will be kept in memory at line 48 as the final solution. Lines 49 to 52 terminate the nested if statements and the for loop started at line 18 respectively.

The time complexity of an algorithm is fundamental to evaluate its practicality in an elastic cloud environment. For PEWOA, it is computed as $O(T(NW * D) + FitFun * NW)$ which is the same as WOAmM. Here, T is the number of iterations, NW is the number of whales in population, D is the dimension of a problem, and FitFun is the cost of evaluating a fitness function. However, WOAmM possesses a greater number of operations in comparison to PEWOA. Although all whale optimization algorithms have a higher inherent complexity, the parallelization in PEWOA makes its execution time much lower.

V. WORKLOAD DATASETS

The proposed PEWOA and other comparative algorithms are assessed using the following two datasets.

A. GOOGLE CLOUD JOBS (GOCJ)

The Google Cluster Traces [36] is a real time log of workloads that ran on Google Borg cluster, comprising 12.5k machines. The trace covers information such as the submission time, scheduling information, and usage of resources. However it does not provide the size of jobs or their deadlines. The trace contains data for 25 million tasks grouped into 650 thousand jobs over a span of 29 days [37]. All the provided data in the trace is normalized and obfuscated to avoid disclosing confidential information. Some jobs (0.003%) are omitted from the trace as they ran on nodes not part of this trace. Some task and job events, 0.013% and 0.0008% respectively, have non empty missing fields. Moreover, data is missing for an average of 0.05% of job/task scheduling events and less than 1% of resource usage records.

It is tedious and infeasible to utilize such an enormous number of tasks in simulations, especially when faced with limited resources. Therefore, the Google Cloud Jobs (GoCJ) dataset is adopted that is derived from Google Cluster Traces 2011 [36] using bootstrapped Monte Carlo (MC) simulation [38]. The GoCJ is a realistic dataset that reflects the workload bahavior of Google Cluster Traces, as asserted by [39], [40], [41], [42], and [43] and analysis of the MapReduce logs from M45 supercomputing cluster by [44].

Instead of randomly choosing values, the original dataset is repeatedly sampled by selecting a single datapoint from the origional dataset in bootstrapping. A list of 50 different sized jobs from the origional dataset is input into the MC boostrapping with equal probability, considering an average computing power of 1000 MIPS for machines. There is a covariance of 2.49 between the origional and average GoCJ datasets. Figure 5 shows the comparison of data distribution for the Original Dataset(O-Dataset) and the 19 GoCJ instances.

In the context of GoCJ, the terms "job" and "task" are used interchangeably, both referring to an independent set of instructions. The distinction between these terms becomes essential when there are dependencies among tasks. Furthermore, it is importatnt to note that the granularity of tasks is higher than that of jobs.

The median of all datasets falls within the range of 870000-970000 MIPS. Similarly the ranges for the first quartile and third quartile are 610000-670000 and 115000-112000 MIPS respectively. The minium and maximum sizes of jobs in all the datasets are also the same. The size of jobs is calculated using the expected time to completion figures in the original dataset, as per Eq. (18).

$$Size\ of\ Job\ (MIPS) = Machine(MIPS)\ x\ ETC \qquad (18)$$

The ratio of different categories of jobs with the correponding ranges of instructions in GoCJ is shown in Figure 4. Medium sized jobs (40%) clearly constitue the highest percentage, followed by large (30%) and small (20%) jobs. Figure 5 illustrates that the extra-large (6%) and huge (4%) sized jobs are the least in proportions respectively.

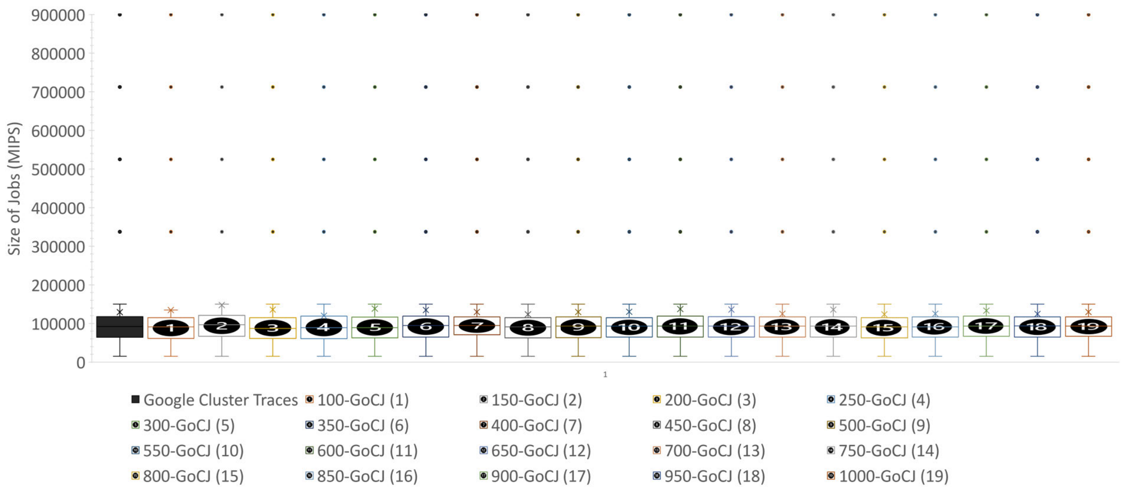

FIGURE 5. Data distribution of google cluster traces and GoCJ instances.

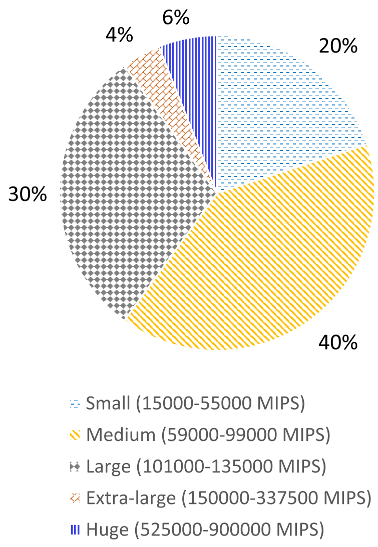

FIGURE 6. Resource mapping distribution in 106 articles included in this study.

B. HETEROGENEOUS COMPUTING SCHEDULING PROBLEM (HCSP)

The Heterogeneous Computing Scheduling Problem dataset [45] is based on the notion of Expected Time to Compute (ETC) for tasks in a heterogeneous environment. The workload assumes each task as an atomic unit and non-preemptive. Additinally, it also assumes that the execution time of a task varies from machine to machine, aiming to minimizes the makespan of tasks. The dataset offers three types of instances (small, medium, and large) based on the size and complexity of tasks and virtual machines. In this paper, the small HCSP instance has been utilized having 1024 tasks and 32 virtual machines.

HCSP uses a notation of c/i_heterogeneity for tasks (hi/lo) and VMs (hi/lo). The "c" and "i" stand for consistency and inconsistency respectively. Heterogeneity for tasks and VMs can be either "hi" or "lo". Low heterogeneity (lo) signifies similar computing resources, while high heterogeneity (ho) indicates a wide range of computing machines. Similarly, the degree of similarity among task execution times is denoted as low heterogeneity and vice versa. For reflecting a realistic scenario, HCSP classifies the workload as consistent (c), inconsistent (i), or semi-consistent (s). Consistency occurs when if a machine executes a task faster than other machines, and it's likely that the same machine will execute other tasks faster in comparision with the rest of machines. In case of inconsistent behavior, a machine may be faster to execute a task, but may not perform similary with other workloads. This category mirros a distributed infrastructure of heterogenous resources with a variety of tasks. A third category is a semi-consistent model, combining characteristics of the first two workloads. Table 3 shows the different configurations of HCSP instances.

TABLE 3. HCSP instances.

Size	Number of Tasks	Number of Machines
Small	1024	32
	2048	64
Medium	4096	128
	8192	256
Large	16384	512
	32768	1024

The ETC matrices are designed using a range based method that incoporates task heterogeneity (R_{TASK}), machine heterogeneity (R_{MACH}), and consistency. Initially a $Tx1$ baseline vector (B) is generated using a uniform distribution of floating point values in the range $[1, R_{TASK}]$. Subsequetly, the rows of $ETC(t_ixm_j)$ matrix are constructed by multiplying the vector B with another uniform random number X (called row multiplier), which falls in the range $[1, R_{MACH}]$. As a

TABLE 4. HCSP workload.

Instances: 4
No. of Tasks: 1024 in each instance
No. of VMs: 32 in each instance

Instances	*Range of instructions*	*Range of VM MIPS*
c_lohi	[23-98121]	[37-999]
i_lohi	[23-98121]	[10-952]
i_hilo	[2-29843]	[1-10]
c_hilo	[2-29843]	[1-10]

result, the ETC matrix comprises the values within the range $[1, R_{TASK} x R_{MACH}]$.

The minimum and maximum values for task heterogeneity are 100 and 3000 respectively, while the corresponding values for machine heterogeneity are 10 and 1000 respectively. The wider range of task heterogeneity (100-3000) compared to machine heterogeniety (10-1000) reflects the greater variablitly in heterogenity for tasks in real world scenarios. For consistent data, the ETC rows are sorted from left to right in descending order creating an ordered dataset. The unsorted ETC matrix constitutes the inconsistent dataset. In the case of semi-consistent data, the even indexed columns' data is extracted for each row, sorted and replaced, while the odd indexed columns remain unchanged. The ETC matrix represents the task dataset, while the participating X table listing the virtual machines. This paper utilizes a small dataset comprising 1024 tasks and 32 machines. In total there are four workload instances as c_lohi, i_lohi, i_hilo, and c_hilo. The size ranges of tasks and virtual machines are provided in Table 4.

VI. EXPERIMENTS AND RESULTS

The experiments are conducted on an Intel Core i7-4790 3.60 GHz processor, equipped with 8 GB of RAM and 1 TB storage. To assess the performance of the proposed Parallel Enhanced Whale Optimization Algorithm (PEWOA), simulations are carried out in CloudSim 3.0.3 using the datasets of Google Cloud Jobs Dataset (GoCJ) and Heterogeneous Computing Scheduling Problem (HCSP). Table 5 details the datacenter configuration for GoCJ according to [18] and HCSP.

The performance of every meta-heuristic algorithm is sensitive to the number of agents and iterations; therefore [18], [19], and [20] will be executed with their proposed number of agents and iterations provided in Table 6. The authors used random generated tasks with a uniform distribution to evaluate benchmark algorithms. For PEWOA and WOAmM, 60 number of agents and 220 number of iterations are selected. With these given numbers, PEWOA provides adequate performance with optimum running time.

While comparing meta-heuristic algorithms, another pertinent aspect to consider is the seed generation (intial population). All algorithms in this paper are using the same seed, except SAEA which undergoes minor changes to simulate location and mapping matrices. The values presented in all experiments indicate the average figures of ten different runs of the algorithms.

TABLE 5. Datacenter configuration for GoCJ and HCSP.

No. of datacenters: 1
PE MIPS: 12000
Hosts: 3 (Quad-core, Hexa-Core, and Octa-Core)
RAM: [512-14436] MB

GoCJ

vm1	400	vm2	440
vm3	600	vm4	800
vm5	900	vm6	1200
vm7	2000	vm8	4000
vm9	10000	vm10	12000

HCSP

No. of VMs: 32 in all four instances

c_lohi	[37-999]	i_lohi	[10-952]
i_hilo	[1-10]	c_hilo	[1-10]

TABLE 6. No. of Agents and iterations.

Algorithm	Agents	Iterations
RMPSO, MRMPSO	32	500
SAEA	100	20
GAMR	50	120
Average	60	213

TABLE 7. Relative increase in average makespan.

Algorithm	WOAmM	RMPSO	MRMPSO	SAEA	GAMR	PEWOA	Total
Average Makespan	5928.78	11104	11148.7	19107	15788.9	2843.4	65921.4
Relative Increase	8.99%	16.84%	16.91%	28.98%	23.95%	**4.31%**	

A. PERFORMANCE ANALYSIS OF PEWOA

1) MAKESPAN

In cloud computing, makespan is the one of the most crucial factors during task scheduling [46]. It represents the completion time of all tasks scheduled on a a set of virtual machines, as expressed by Eq. (7) [10]. A lower makespan makes a cloud server efficient to execute workloads swiftly. In Figure 7, the makespan of all algorithms increases with the increasing number of tasks. However, the proposed PEWOA performs considerably well to keep the makespan low through the optimal assignment of virtual machines to incoming tasks. The second best makespan figures are shown by WOAmM [17], an enhanced whale optimiation algorithm. The bahavior of SAEA and GAMR is similar, while RMPSO and MRMPSO exhibit nearly identical makespans, as MRMPSO is a parallel version of RMPSO. The relative increase in average makespan for all algorithms is illustrated in Table 7.

PEWOA demonstrates the ability to handle an increasing number of tasks with a relatively modest increase in makespan. In Table 7, PEWOA indicates the least increase (4.31%) in average makespan over 19 GoCJ instances, showcasing better scalability. WOAmM possess the second

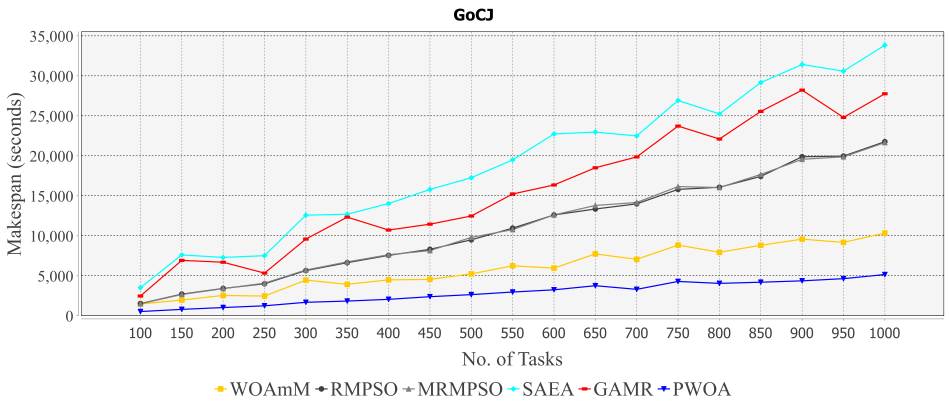

FIGURE 7. Makespan of tasks on various GoCJ instances.

least increase (8.99%) in average makespan. RMPSO and MRMPSO remain the third-best algorithms, with increasing makespan figures of 16.84% and 16.91% respectively. The worst figures (28.98% and 23.95%) are exhibited by SAEA and GAMR respectively, indicating thier minimal ability to schedule a growing number of tasks. It is pertinent to mention that the makespan figures of SAEA and GAMR show unpredicted highs and lows with various GoCJ instances. The average makespan values for all algorithms are provided in Table 8. Cloud service providers often have SLAs with users specifying the maximum time frame for executing their workload. So, a minimum makespan helps companies to provide results within the agreed upon time duration.

On HCSP workload, a similar behavior is observed with reduced makespan as shown in Figure 8. For a consistent dataset comprising of low heterogeneous tasks and highly heterogeneous vms, the makespan of PEWOA is the lowest as compared to WOAmM, RMPSO, MRMPSO, SAEA, and GAMR, but with a relatively less margin especially in comparision to GAMR. The smallar bars in the graph for c_lohi result from the higher MIPS capacities of vms compared to the smaller size of tasks. There is a small difference between WOAmM and PEWOA for c_lohi, but the difference in makespan increases with the increasing complexity of tasks and vms in i_hilo. For i_lohi, the makesapan of both PEWOA and WOAmM remain unaffected by the low heterogeneous tasks and high heterogeneous vms unlike other algorithms. GAMR struggles with the inconsistent behavior of i_lohi. For i_hilo, where the ratio of the size of tasks to vms is the largest, the makespan of all algorithms is the highest. Despite the parallelism in MRMPSO, RMPSO and MRMPSO have the same makespan figures, while PEWOA still outperforms others. Although SAEA and GAMR are parallelized algorithms utilizing multiple subpopulations and concurrent fitness evaluation respectively, yet they show the worst results. The SAEA is dominant over GAMR by effectively managing the high heterogenity of tasks. In case of c_hilo, the results are similar to c_lohi, but the different magnitude of the bars.

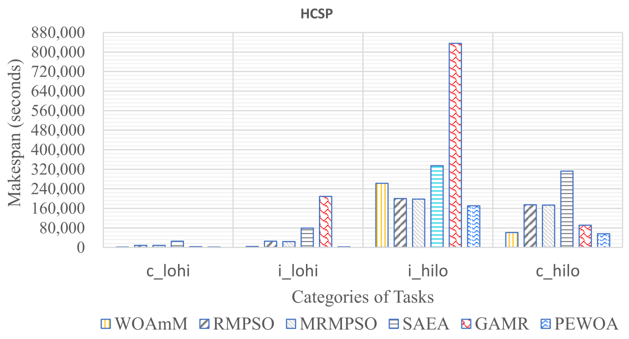

FIGURE 8. Makespan of tasks on four HCSP instances.

PEWOA continues to provide a better makespan with high task heterogeneity. WOAmM shows minimum makespan figures after PEWOA. The results of both PSO variants are identical, but the scheduling behavior of both SAEA and GAMR differs, with GAMR performing better than SAEA. The average makespan values for all algorithms are provided in Table 13.

2) RESOURCE UTILIZATION

A higher resource utilization enables the use of fewer resources, resulting in savings and a number of advantages. Cloud providers strive to use fewer resources while meeting QoS using Eq. (8) [10]. Figure 9 clearly indicates that the proposed PEWOA achieves the highest resource utilization compared to the benchmark algorithms. WOAmM has better resource utilization than RMPSO, and MRMPSO, while GAMR displays better resource utilization than SAEA. Despite implementing a migration policy, SAEA has the lowest resource utilization among all algorithms. The average values of resource utilization for all algorithms are presented in Table 9.

In Figure 10, for c_lohi, the resource utilization of PEWOA is significantly higher than others due to the execution of parallel threads and enhancements, while RMPSO, MRMPSO, and GAMR show nearly identical utilization of resources.

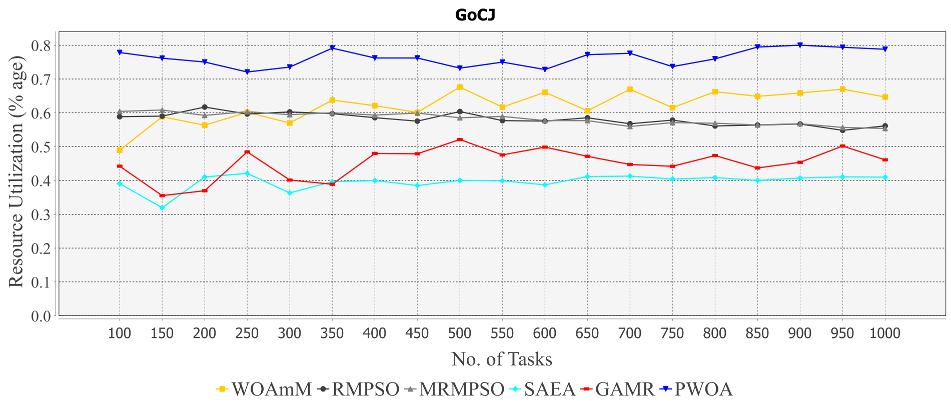

FIGURE 9. Resource utilization on various GoCJ instances.

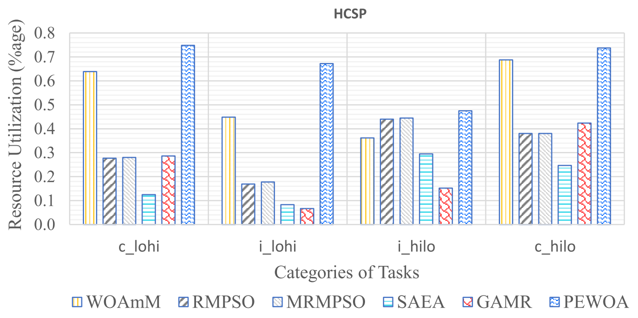

FIGURE 10. Resource utilization on four HCSP instances.

WOAMm has consistently better performance in all the tests after PEWOA. The parallelization of multiple sub-populations in SAEA does not lead to improved results. It is evident that the inconsistent nature of i_lohi has negatively the resource utilization of all algorithms, with the least impact on SAEA. Similarly, MRMPSO and PEWOA show identical reductions in resource utilization. Better resource utilization reduces the idle time of individual resources in the cloud. Idle resources represent wasted capacity that could be used for executing tasks. The ability to scale with dynamic workloads also depends on the proper resource utilization of avaiable resources. PEWOA manage to provide better performance on varying workloads due to its improved resource utilization. Similarly, resource utilization has multifaceted effects on factors such as cost and energy consumption; however, these aspects are beyond the scope of this study. The average values of resource utilization for all algorithms are provided in Table 17.

3) THROUGHPUT

Throughput is a key indicator of the overall efficiency of a cloud. It is defined as the number of tasks completed per unit time, expressed by Eq. (9) [11]. A system with high throughput makes efficient utilization of resources. Figure 11 illustrates the highest throughput achieved by PEWOA, primarily attributed to running multiple threads and the modified encircling move and logrithmic spiral mechanism. There is a notable difference between the throughput of the proposed algorithm and the rest of benchmark algorithms. The average values of throughput for all algorithms are provided in Table 11.

On HCSP, Figure 12 depicts the least difference in throughput among all algorithms for i_hilo, followed by c_hilo. The number of tasks executed per second is the highest for all algorithms in the case of c_lohi because all algorithms perform well with a consistent set of tasks and vms. However, for i_lohi, the throughput of all algorithms is negatively affected, particularly GAMR. The average values of throughput for all algorithms are provided in Table 15.

4) RESPONSE TIME

The time a virtual machine takes to start executing a mapped task after task scheduling is called response time. A lower response time indicates a higher level of productivity and performance. For task scheduling in a virtualized environment, Eq. (10) computes the average response time. In Figure 13, the average response time of all benchmark algorithms is the same, but our proposed PEWOA shows relatively the minimum figures for 16 out of 19 GoCJ instances. The average values of response time for all algorithms are provided in Table 10.

Unlike other optimizaion metrics, the response time of all HCSP instances is nearly the same for all algorithms, execpt for c_lohi where GAMR and PEWOA show better response time. For i_hilo, the response time of all algorithms is identical, while for i_lohi and c_hilo, both WOAmM and PEWOA indicate minor improvements over the benchmark algorithms as depicted in Figure 14. The average values of response time for all algorithms are provided in Table 16.

5) EXECUTION TIME

The execution time of a meta-heuristic algorithm is also crucial in a scalable environment. It is measured as the difference of starting time and complete execution of an algorithm. In Figure 15, WOAMm exhibit the worst

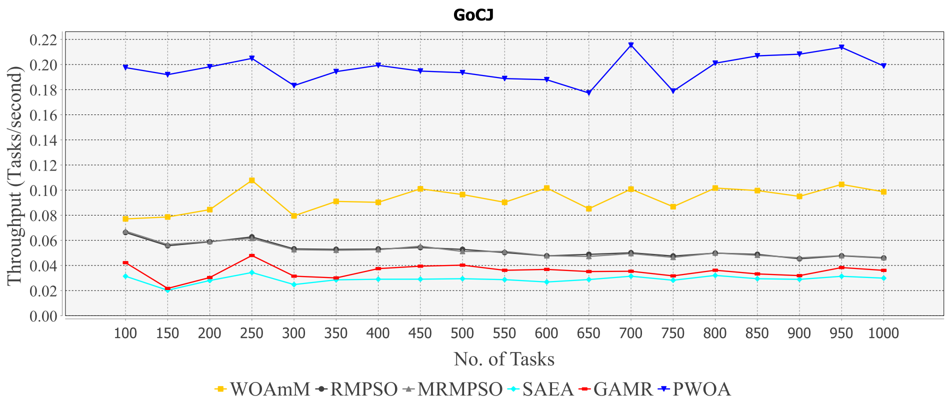

FIGURE 11. Throughput on various GoCJ instances.

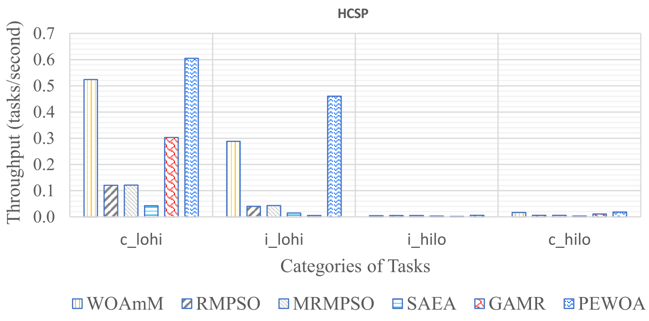

FIGURE 12. Throughput on four HCSP instances.

execution time due to the internal complexity of whale optimization algorithm followed by RMPSO. However, the parallelized variant of RMPSO shows a considerably lower execution time. The proposed PEWOA has a lower execution time than RMPSO and MRMPSO, even though it involves more computations. However, it is still inferior to SAEA and GAMR. SAEA has low time complexity due to its simple internal structure and the utilization of multiple sub-population of squirrels. GAMR has the lowest execution time, attributed to its undemanding implementation of crossover and mutation. The average values of execution time for different algorithms are provided in Table 12.

Similarly, in Figure 16, GAMR exhibits the lowest execution time with a mutation probability of less than 0.7. PEWOA has the second best execution time because of multithreading. The migration of squirrels in SAEA is complicated, possibly contributing to its higher execution time. Notably, there is a significant difference observed between RMPSO and MRMPSO for the first time. The latter uses a master-slave parallelization model, reducing the overall execution time as compared to the former. PEWOA shows better execution time on HCSP as compared to GoCJ, where it has the second lowest execution time. WOAMm has the highest execution time on HCSP workload as well. A scheduling algorithm with lower execution time enables the handling of diverse workloads effectively. The average values of execution time for all algorithms are provided in Table 17.

TABLE 8. Average makespan on GoCJ.

Dataset	WOAmM	RMPSO	MRMPSO	SAEA	GAMR	PEWOA
GoCJ_100	1463.487	1514.845	1487.443	3508.848	2461.698	**521.030**
GoCJ_150	1958.589	2699.207	2652.593	7599.550	6920.225	**789.909**
GoCJ_200	2532.894	3409.200	3386.003	7287.144	6687.973	**1021.085**
GoCJ_250	2456.095	3990.430	4065.302	7506.685	5328.423	**1235.832**
GoCJ_300	4447.292	5642.228	5714.017	12572.382	9581.120	**1675.152**
GoCJ_350	3927.503	6630.561	6725.675	12693.636	12327.779	**1829.566**
GoCJ_400	4477.313	7543.867	7620.459	14020.706	10711.936	**2045.443**
GoCJ_450	4544.007	8298.166	8144.292	15788.495	11445.531	**2377.799**
GoCJ_500	5226.545	9488.949	9792.555	17252.446	12467.720	**2635.601**
GoCJ_550	6236.28	10950.621	10767.388	19489.488	15222.441	**2959.369**
GoCJ_600	5953.176	12600.051	12600.732	22726.706	16347.720	**3236.624**
GoCJ_650	7729.039	13344.128	13787.532	22964.222	18499.786	**3749.641**
GoCJ_700	7049.795	13987.357	14159.726	22485.678	19856.651	**3296.320**
GoCJ_750	8830.929	15797.220	16148.089	26915.205	23716.501	**4282.304**
GoCJ_800	7930.519	16062.512	16029.629	25233.323	22098.114	**4042.842**
GoCJ_850	8806.476	17419.829	17668.335	29160.245	25549.115	**4187.579**
GoCJ_900	9575.947	19878.048	19560.945	31415.305	28208.218	**4358.422**
GoCJ_950	9170.209	19962.925	19868.155	30586.949	24800.835	**4632.987**
GoCJ_1000	10330.64	21763.657	21645.943	33831.836	27757.346	**5146.297**

The proposed algorithm exhibits a considerably lower makespan than WOAmM, RMPSO, MRMPSO, SAEA, and GAMR. It also demonstrates superior utilization of virtual resources and higher throughput for both workload instances of GoCJ and HCSP. The execution time of PEWOA has been reduced using multi-threading to schedule tasks efficiently. The algorithm consistently achieves lower makespan, higher resource utilization, and greater throughput for various workload instances, demonstrating better scalability. This scalability makes PEWOA well-suited for an elastic cloud infrastructure, where the scale of workloads continuously grows and shrinks.

B. STATISTICAL TESTS

1) STANDARD DEVIATION

It represents the variation or dispersion in a given set of values. Table 18 illustrates that PEWOA has the most consistent makespan followed by WOAmM. Although the PEWOA average resource utilization is better among all, MRMPSO shows more consistent figures. Similary, despite

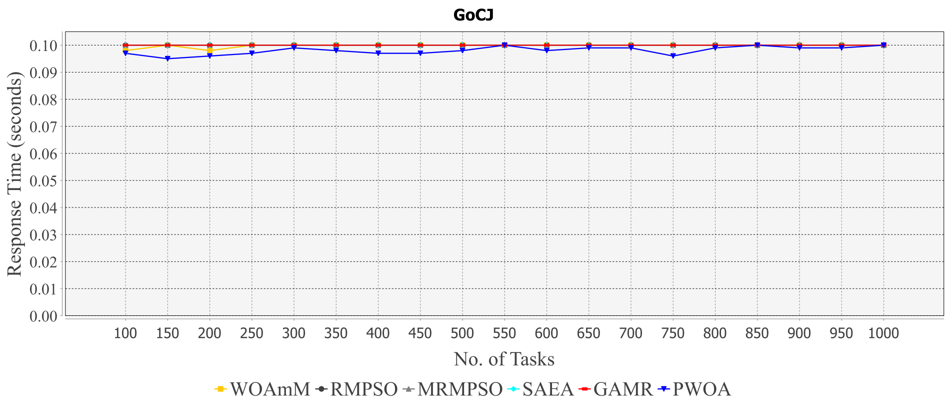

FIGURE 13. Task Response time on various GoCJ instances.

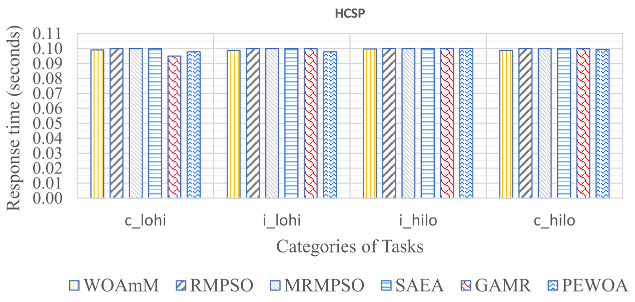

FIGURE 14. Task Response time on various GoCJ instances.

TABLE 9. Average resource utilization on GoCJ.

Dataset	WOAmM	RMPSO	MRMPSO	SAEA	GAMR	PEWOA
GoCJ_100	0.489	0.589	0.604	0.391	0.442	**0.778**
GoCJ_150	0.588	0.590	0.608	0.319	0.355	**0.762**
GoCJ_200	0.563	0.617	0.592	0.410	0.370	**0.750**
GoCJ_250	0.602	0.596	0.603	0.421	0.484	**0.721**
GoCJ_300	0.57	0.603	0.594	0.363	0.401	**0.735**
GoCJ_350	0.638	0.597	0.600	0.396	0.389	**0.791**
GoCJ_400	0.621	0.586	0.593	0.400	0.480	**0.762**
GoCJ_450	0.601	0.575	0.599	0.385	0.479	**0.762**
GoCJ_500	0.676	0.604	0.585	0.400	0.521	**0.732**
GoCJ_550	0.617	0.577	0.589	0.399	0.476	**0.750**
GoCJ_600	0.661	0.576	0.577	0.387	0.499	**0.728**
GoCJ_650	0.606	0.585	0.577	0.411	0.471	**0.772**
GoCJ_700	0.67	0.568	0.560	0.413	0.447	**0.776**
GoCJ_750	0.615	0.578	0.571	0.404	0.442	**0.737**
GoCJ_800	0.662	0.561	0.569	0.408	0.474	**0.760**
GoCJ_850	0.649	0.564	0.564	0.400	0.437	**0.794**
GoCJ_900	0.659	0.567	0.568	0.407	0.454	**0.800**
GoCJ_950	0.67	0.548	0.557	0.411	0.502	**0.794**
GoCJ_1000	0.647	0.561	0.554	0.410	0.461	**0.788**

TABLE 10. Average response time on GoCJ.

Dataset	WOAmM	RMPSO	MRMPSO	SAEA	GAMR	PEWOA
GoCJ_100	9.80E-02	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.7E-02**
GoCJ_150	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.5E-02**
GoCJ_200	9.80E-02	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.6E-02**
GoCJ_250	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.7E-02**
GoCJ_300	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.9E-02**
GoCJ_350	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.8E-02**
GoCJ_400	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.7E-02**
GoCJ_450	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.7E-02**
GoCJ_500	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.8E-02**
GoCJ_550	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01
GoCJ_600	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.8E-02**
GoCJ_650	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.9E-02**
GoCJ_700	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.9E-02**
GoCJ_750	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.6E-02**
GoCJ_800	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.9E-02**
GoCJ_850	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01
GoCJ_900	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.9E-02**
GoCJ_950	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	**9.9E-02**
GoCJ_1000	1.00E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01	1.0E-01

the PEWOA's impressive throughput, SAEA shows consistent behavior for the number of tasks completed per unit time. The response time of SAEA and GAMR display the least variation. Additionally, the execution time of GAMR is the lowest among all algoirthms and is least affected by the increasing number of GoCJ tasks.

The standard deviation figures (in Table 19) on HCSP shows a similar pattern with minor changes for MRMPSO and SAEA, where the latter has a lower range of resource utilization values than the former. The response time for RMPSO, MRMPSO, and SAEA does not show any variation across the four HCSP instances.

2) FRIEDMAN TEST

The Friedman test [47] is a non-parametric statistical test developed by Milton Friedman. It is used to detect changes in various techniques applied on a particular dataset. To illustrate the behavior of PEWOA againts other algorithms, the Friedman test is performed on all instances of GoCJ and

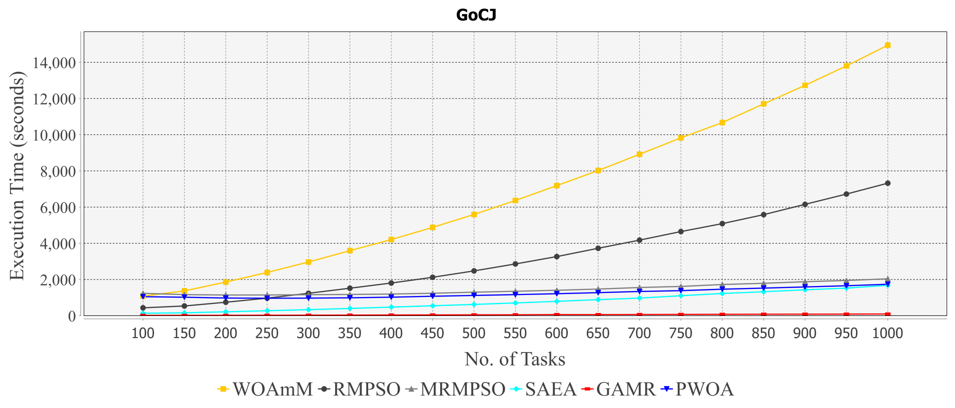

FIGURE 15. Execution Time on various GoCJ instances.

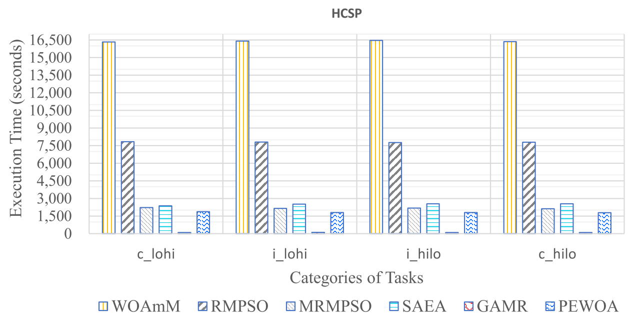

FIGURE 16. Execution Time on four HCSP instances.

TABLE 11. Average throughput on GoCJ.

Dataset	WOAmM	RMPSO	MRMPSO	SAEA	GAMR	PEWOA
GoCJ_100	0.077	0.066	0.067	0.031	0.042	**0.198**
GoCJ_150	0.079	0.056	0.057	0.020	0.022	**0.192**
GoCJ_200	0.084	0.059	0.059	0.028	0.030	**0.198**
GoCJ_250	0.108	0.063	0.062	0.034	0.048	**0.205**
GoCJ_300	0.079	0.053	0.053	0.025	0.031	**0.183**
GoCJ_350	0.091	0.053	0.052	0.029	0.030	**0.194**
GoCJ_400	0.09	0.053	0.053	0.029	0.037	**0.199**
GoCJ_450	0.101	0.054	0.055	0.029	0.039	**0.195**
GoCJ_500	0.096	0.053	0.051	0.029	0.040	**0.193**
GoCJ_550	0.09	0.050	0.051	0.029	0.036	**0.189**
GoCJ_600	0.102	0.048	0.048	0.027	0.037	**0.188**
GoCJ_650	0.085	0.049	0.047	0.029	0.035	**0.177**
GoCJ_700	0.101	0.050	0.049	0.031	0.035	**0.215**
GoCJ_750	0.087	0.048	0.046	0.028	0.032	**0.179**
GoCJ_800	0.102	0.050	0.050	0.032	0.036	**0.201**
GoCJ_850	0.1	0.049	0.048	0.029	0.033	**0.207**
GoCJ_900	0.095	0.045	0.046	0.029	0.032	**0.208**
GoCJ_950	0.105	0.048	0.048	0.031	0.038	**0.214**
GoCJ_1000	0.099	0.046	0.046	0.030	0.036	**0.199**

HCSP. The values for the test statistic also known as chi square (χ^2) for makespan, resource utilization, throughput, response time, and exeuction time are provided in Table 20. Based on the given values, the null hypothesis (H_0) (that there is no difference among the given algorithms) is rejected for both GoCJ and HCSP datasets. The computed chi square values are significantly larger than the corresponding critical values at 5 degrees of freedom (df).

TABLE 12. Average execution time on GoCJ.

Dataset	WOAmM	RMPSO	MRMPSO	SAEA	GAMR	PEWOA
GoCJ_100	1081.400	434.900	1236.900	137.200	**19.600**	1050.900
GoCJ_150	1370.100	536.400	1164.700	159.400	**17.800**	1020.400
GoCJ_200	1858.700	743.600	1142.500	214.800	**21.000**	975.700
GoCJ_250	2388.500	969.700	1144.500	273.700	**24.800**	964.400
GoCJ_300	2966.700	1238.200	1160.500	333.700	**28.700**	970.700
GoCJ_350	3594.200	1519.000	1173.200	401.600	**33.500**	989.200
GoCJ_400	4207.800	1805.300	1196.900	475.300	**38.600**	1023.100
GoCJ_450	4879.500	2122.300	1240.300	547.900	**43.000**	1074.000
GoCJ_500	5592.800	2475.500	1298.400	623.600	**47.100**	1119.500
GoCJ_550	6367.500	2859.400	1349.900	704.000	**52.300**	1164.400
GoCJ_600	7187.000	3266.300	1405.400	793.000	**62.700**	1214.000
GoCJ_650	8021.200	3724.900	1471.800	887.200	**60.900**	1273.600
GoCJ_700	8919.400	4176.300	1557.200	971.800	**65.500**	1338.300
GoCJ_750	9827.200	4648.500	1617.000	1103.500	**70.500**	1383.800
GoCJ_800	10670.900	5087.100	1723.300	1227.700	**76.200**	1463.700
GoCJ_850	11704.200	5583.200	1791.400	1322.900	**82.200**	1524.500
GoCJ_900	12729.100	6152.400	1876.600	1431.600	**85.000**	1588.000
GoCJ_950	13802.400	6721.100	1946.700	1524.000	**88.300**	1654.800
GoCJ_1000	14946.200	7321.700	2038.900	1679.500	**93.700**	1737.000

TABLE 13. Average makespan figures on HCSP.

Algorithm	c_lohi	i_lohi	i_hilo	c_hilo
WOAmM	1959.922	4025.723	262788.750	61083.419
RMPSO	8545.611	25450.188	200035.233	175057.815
MRMPSO	8474.626	24076.824	197722.970	173620.533
SAEA	25276.431	79216.640	334674.280	312228.480
GAMR	3381.288	209298.646	835479.187	91269.756
PEWOA	**1693.166**	**2236.068**	**170632.216**	**55905.593**

TABLE 14. Average resource utilization figures on HCSP.

Algorithm	c_lohi	i_lohi	i_hilo	c_hilo
WOAmM	0.638	0.448	0.362	0.687
RMPSO	0.277	0.169	0.440	0.381
MRMPSO	0.280	0.178	0.444	0.380
SAEA	0.125	0.083	0.295	0.246
GAMR	0.286	0.067	0.152	0.424
PEWOA	**0.748**	**0.672**	**0.475**	**0.737**

3) WILCOXON TEST

The Wilcoxon Signed-rank test [48] is another non-parametric test developed by Frank Wilcoxon. It is employed to identify any significant difference between two pairs of data. This test is also performed on every benchmark

TABLE 15. Average throughput figures on HCSP.

Algorithm	c_lohi	i_lohi	i_hilo	c_hilo
WOAmM	0.523	0.288	0.004	0.017
RMPSO	0.120	0.040	0.005	0.006
MRMPSO	0.121	0.043	0.005	0.006
SAEA	0.043	0.015	0.003	0.003
GAMR	0.303	0.005	0.001	0.011
PEWOA	**0.605**	**0.461**	**0.006**	**0.018**

TABLE 16. Average response time figures on HCSP.

Algorithm	c_lohi	i_lohi	i_hilo	c_hilo
WOAmM	0.099	0.099	0.100	0.099
RMPSO	0.100	0.100	0.100	0.100
MRMPSO	0.100	0.100	0.100	0.100
SAEA	0.100	0.100	0.100	0.100
GAMR	**0.095**	0.099	**0.099**	0.100
PEWOA	0.098	**0.098**	0.100	**0.099**

TABLE 17. Average execution time figures on HCSP.

Algorithm	c_lohi	i_lohi	i_hilo	c_hilo
WOAmM	16336.700	16398.300	16457.300	16354.100
RMPSO	7831.200	7810.600	7757.500	7791.200
MRMPSO	2227.500	2155.800	2177.700	2126.900
SAEA	2368.900	2519.500	2542.200	2547.800
GAMR	**102.400**	**110.100**	**100.300**	**101.500**
PEWOA	1872.500	1797.400	1803.400	1788.100

TABLE 18. Standard deviation of various optimization metrics on GoCJ.

Algorithm	Makespan	Resource Utiliza-tion	Throughput	Response Time	Execution Time
WOAmM	2773.8726	0.0469	0.0093	6.31E-04	4393.9397
RMPSO	6309.5726	0.0176	0.0055	1.04E-16	2196.3077
MRMPSO	6302.3985	**0.0175**	0.0057	9.58E-17	300.4388
SAEA	9169.2995	0.0227	**0.0029**	**9.03E-17**	490.4045
GAMR	8013.3373	0.0451	0.0055	**9.03E-17**	**25.1911**
PEWOA	**1416.4970**	0.0246	0.0106	0.0014	254.2206

TABLE 19. Standard deviation of various optimization metrics on all HCSP instances.

Algorithm	Makespan	Resource Utiliza-tion	Throughput	Response Time	Execution Time
WOAmM	123298.5734	0.1544	0.2477	0.0004	53.7948
RMPSO	99233.1454	0.1191	0.0541	**0**	31.3418
MRMPSO	98500.7474	0.1167	0.0545	**0**	42.4624
SAEA	158386.3922	**0.0997**	**0.0185**	**0**	84.6886
GAMR	376651.2335	0.1563	0.1486	0.0022	**4.4342**
PEWOA	**79519.1606**	0.1266	0.3063	0.0011	38.6164

TABLE 20. χ^2 values for GoCJ and HCSP.

Metrics	GoCJ	HCSP
Makespan	92.29	14.71
Resource Utilization	85.56	14.28
Throughput	89.47	14.28
Response Time	29.17	8.92
Execution Time	89.40	20
Critical values at df = 5	9.432 for $\alpha = 5\%$, 12.88 for $\alpha = 1\%$	8.800 for $\alpha = 5\%$, 11.20 for $\alpha = 1\%$

algorithm against PEWOA for any significant differences in makespan, resource utilization, response time, and execution time. With four instances in HCSP, the Wilcoxon test is unable to identify any difference. Therefore, instead of the average values over ten separate runs, all records have been used, making the total number of records 40 to calculate the Wilcoxon test successfully. The values of Wilcoxon Statistic (W) for both GoCJ and HCSP are provided in Table 21.

TABLE 21. Wilcoxon statistic values for GoCJ and HCSP.

Metrics	GoCJ	HCSP
Makespan	0	0
Resource Utilization	0	*PEWOA-MRMPSO=3, PEWOA-WOAmM=182, Others=0*
Throughput	0	0
Response Time	*PEWOA-WOAmM=17.5, Others=0*	325-381
Execution Time	*PEWOA-RMPSO=12, Others=0*	*PEWOA-MRMPSO=1, PEWOA-SAEA=1, Others=0*
Critical values:	46 for $\alpha = 5\%$, 32 for $\alpha = 1\%$	264 for $\alpha = 5\%$, 220 for $\alpha = 1\%$

For GoCJ, the critical value of Wilcoxon‘s statistic is 46, therefore H_0 is rejected for PEWOA against all algorithms. Similarly, on HCSP workload, H_0 is rejected except for response time because the Wilcoxon's statistic values are greater than the critical value of 264. Hence, there is not a significant difference in the response time of PEWOA in comparison to WOAmM, RMPSO, MRMPSO, SAEA, and GAMR.

VII. CONCLUSION

Task scheduling poses a significant NP-hard problem in cloud computing, impacting the efficiency and resource utilization of cloud datacenters. Efficient scheduling is a prime factor for better quality of service. Heuristic algorithms such as DRALBA, OG-RADL and SG-PBFS etc. schedule tasks on a given set of resources by either prioritizing tasks or ranking resources that hamper the resource utilization of a system and ultimately lower makespan. However, a meta-heuristic algorithm treats all tasks and resources impartially. However, the right balance between global search and local search in a meta-heuristic algorithm is exigent to provide an optimal result. In response to this, a Parallel Enhanced Whale Optimization Algorithm (PEWOA) is proposed for scheduling of independent tasks on heterogeneous virtual machines in the cloud. PEWOA incorporates parallelization, an updated encircling maneuver and a bubble net attacking mechanism to enhance the solution diversity, avoid local optima, and improve convergence. The enhanced encircling maneuver and bubble net attacking mechanism optimized the solution quality by hitting the right balance between exploration and exploitation at the right time. Despite the internal complexity, parallelization reduced its execution time. Extensive simulations demonstrate that PEWOA minimizes makespan, response time, and increases resource utilization and throughput against WOAmM, RMPSO, MRMPSO, SAEA, and GAMR. The proposed PEWOA provides superior scalability and efficient task scheduling across 19 workload instances of GoCJ. In the case of HCSP with four workload instances, PEWOA maintains similar performance figures while addressing various heterogeneity levels among tasks and virtual machines. Statistical tests, including Standard Deviation, Friedman test, and Wilcoxon test, confirm the significance of the results. In future, it is planned to further improve the algorithm, specifically tailored for task scheduling on fog computing environments.

REFERENCES

[1] J. K. Konjaang and L. Xu, "Meta-heuristic approaches for effective scheduling in infrastructure as a service cloud: A systematic review," *J. Netw. Syst. Manage.*, vol. 29, no. 2, pp. 1–57, Apr. 2021.

[2] M. K. Hussein, M. H. Mousa, and M. A. Alqarni, "A placement architecture for a container as a service (CaaS) in a cloud environment," *J. Cloud Comput.*, vol. 8, no. 1, pp. 1–15, Dec. 2019.

[3] A. Almadhor, A. Alharbi, A. M. Alshamrani, W. Alosaimi, and H. Alyami, "A new offloading method in the green mobile cloud computing based on a hybrid meta-heuristic algorithm," *Sustain. Comput. Informat. Syst.*, vol. 36, Dec. 2022, Art. no. 100812.

[4] Z. Tong, F. Ye, B. Liu, J. Cai, and J. Mei, "DDQN-TS: A novel bi-objective intelligent scheduling algorithm in the cloud environment," *Neurocomputing*, vol. 455, pp. 419–430, Sep. 2021.

[5] Z. A. Khan, I. Ullah, M. Ibrahim, M. Fayaz, A. Aljarbouh, and M. S. Qureshi, "Virtualization based efficient service matching and discovery in Internet of Things," *Electronics*, vol. 9, no. 6, p. 1007, Jun. 2020.

[6] M. S. Ajmal, Z. Iqbal, F. Z. Khan, M. Ahmad, I. Ahmad, and B. B. Gupta, "Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers," *Comput. Electr. Eng.*, vol. 95, Oct. 2021, Art. no. 107419.

[7] A. A. Khan, M. Zakarya, R. Khan, I. U. Rahman, M. Khan, and A. U. R. Khan, "An energy, performance efficient resource consolidation scheme for heterogeneous cloud datacenters," *J. Netw. Comput. Appl.*, vol. 150, Jan. 2020, Art. no. 102497.

[8] H. Cao, S. Wu, G. S. Aujla, Q. Wang, L. Yang, and H. Zhu, "Dynamic embedding and quality of service-driven adjustment for cloud networks," *IEEE Trans. Ind. Informat.*, vol. 16, no. 2, pp. 1406–1416, Feb. 2020.

[9] A. Hussain, M. Aleem, M. A. Iqbal, and M. A. Islam, "SLA-RALBA: Cost-efficient and resource-aware load balancing algorithm for cloud computing," *J. Supercomput.*, vol. 75, no. 10, pp. 6777–6803, Oct. 2019.

[10] S. Nabi and M. Ahmed, "OG-RADL: Overall performance-based resource-aware dynamic load-balancer for deadline constrained cloud tasks," *J. Supercomput.*, vol. 77, no. 7, pp. 7476–7508, Jul. 2021.

[11] S. Nabi, M. Ibrahim, and J. M. Jimenez, "DRALBA: Dynamic and resource aware load balanced scheduling approach for cloud computing," *IEEE Access*, vol. 9, pp. 61283–61297, 2021.

[12] Ä. M. Eligüzel and E. Özceylan, "Application of an improved discrete crow search algorithm with local search and elitism on a humanitarian relief case," *Artif. Intell. Rev.*, vol. 54, no. 6, pp. 4591–4617, Aug. 2021.

[13] Y. Wang, J. Zhao, K. Jiang, Q. Zhou, Z. Kang, C. Chen, and H. Zhang, "Prediction of TBM operation parameters using machine learning models based on BPSO," *Adv. Eng. Informat.*, vol. 56, Apr. 2023, Art. no. 101955.

[14] S. A. Murad, Z. R. M Azmi, F. J. Brishti, M. Saib, and A. K. Bairagi, "Priority based fair scheduling: Enhancing efficiency in cloud job distribution," in *Proc. IEEE 8th Int. Conf. Softw. Eng. Comput. Syst. (ICSECS)*, Aug. 2023, pp. 170–175.

[15] S. A. Murad, Z. R. M. Azmi, A. J. M. Muzahid, M. K. B. Bhuiyan, M. Saib, N. Rahimi, N. J. Prottasha, and A. K. Bairagi, "SG-PBFS: Shortest gap-priority based fair scheduling technique for job scheduling in cloud environment," *Future Gener. Comput. Syst.*, vol. 150, pp. 232–242, Jan. 2024.

[16] Z. A. Khan, I. A. Aziz, N. A. B. Osman, and I. Ullah, "A review on task scheduling techniques in cloud and fog computing: Taxonomy, tools, open issues, challenges, and future directions," *IEEE Access*, vol. 11, pp. 143417–143445, 2023.

[17] S. Chakraborty, A. K. Saha, S. Sharma, S. Mirjalili, and R. Chakraborty, "A novel enhanced whale optimization algorithm for global optimization," *Comput. Ind. Eng.*, vol. 153, Mar. 2021, Art. no. 107086.

[18] X. Tang, C. Shi, T. Deng, Z. Wu, and L. Yang, "Parallel random matrix particle swarm optimization scheduling algorithms with budget constraints on cloud computing systems," *Appl. Soft Comput.*, vol. 113, Dec. 2021, Art. no. 107914.

[19] B. M. H. Zade, N. Mansouri, and M. M. Javidi, "SAEA: A security-aware and energy-aware task scheduling strategy by parallel squirrel search algorithm in cloud environment," *Expert Syst. Appl.*, vol. 176, Aug. 2021, Art. no. 114915.

[20] Z. Peng, P. Pirozmand, M. Motevalli, and A. Esmaeili, "Genetic algorithm-based task scheduling in cloud computing using MapReduce framework," *Math. Problems Eng.*, vol. 2022, pp. 1–11, Sep. 2022.

[21] M. A. Elaziz, S. Xiong, K. P. N. Jayasena, and L. Li, "Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution," *Knowl.-Based Syst.*, vol. 169, pp. 39–52, Apr. 2019.

[22] S. Ben Alla, H. Ben Alla, A. Touhafi, and A. Ezzati, "An efficient energy-aware tasks scheduling with deadline-constrained in cloud computing," *Computers*, vol. 8, no. 2, p. 46, Jun. 2019.

[23] M. S. Sanaj and P. M. J. Prathap, "Nature inspired chaotic squirrel search algorithm (CSSA) for multi objective task scheduling in an IAAS cloud computing atmosphere," *Eng. Sci. Technol., Int. J.*, vol. 23, no. 4, pp. 891–902, Aug. 2020.

[24] M. R. Shirani and F. Safi-Esfahani, "Dynamic scheduling of tasks in cloud computing applying dragonfly algorithm, biogeography-based optimization algorithm and Mexican hat wavelet," *J. Supercomput.*, vol. 77, no. 2, pp. 1214–1272, Feb. 2021.

[25] J. Prassanna and N. Venkataraman, "Adaptive regressive holt–winters workload prediction and firefly optimized lottery scheduling for load balancing in cloud," *Wireless Netw.*, vol. 27, no. 8, pp. 5597–5615, Nov. 2021.

[26] M. A. Elaziz and I. Attiya, "An improved Henry gas solubility optimization algorithm for task scheduling in cloud computing," *Artif. Intell. Rev.*, vol. 54, no. 5, pp. 3599–3637, 2021.

[27] A. Tarafdar, M. Debnath, S. Khatua, and R. K. Das, "Energy and makespan aware scheduling of deadline sensitive tasks in the cloud environment," *J. Grid Comput.*, vol. 19, no. 2, pp. 1–25, Jun. 2021.

[28] M. Kaushik, P. Jharashree, and K. Santosh, "A dynamic load scheduling in IaaS cloud using binary Jaya algorithm," *J. King Saud Univ.*, vol. 12, pp. 2–4, Sep. 2020.

[29] M. Nanjappan, G. Natesan, and P. Krishnadoss, "HFTO: Hybrid firebug tunicate optimizer for fault tolerance and dynamic task scheduling in cloud computing," *Wireless Pers. Commun.*, vol. 129, no. 1, pp. 323–344, Mar. 2023.

[30] P. Banerjee, S. Roy, A. Sinha, M. M. Hassan, S. Burje, A. Agrawal, A. K. Bairagi, S. Alshathri, and W. El-Shafai, "MTD-DHJS: Makespan-optimized task scheduling algorithm for cloud computing with dynamic computational time prediction," *IEEE Access*, vol. 11, pp. 105578–105618, 2023.

[31] S. P. Praveen, H. Ghasempoor, N. Shahabi, and F. Izanloo, "A hybrid gravitational emulation local search-based algorithm for task scheduling in cloud computing," *Math. Problems Eng.*, vol. 2023, pp. 1–9, Feb. 2023.

[32] J. Yan, Y. Lu, L. Chen, S. Qin, Y. Fang, Q. Lin, T. Moscibroda, S. Rajmohan, and D. Zhang, "Solving the batch stochastic bin packing problem in cloud: A chance-constrained optimization approach," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2022, pp. 2169–2179.

[33] M. Ewais and P. Chow, "Disaggregated memory in the datacenter: A survey," *IEEE Access*, vol. 11, pp. 20688–20712, 2023.

[34] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016, doi: 10.1016/j.advengsoft.2016.01.008.

[35] L. S. Nair, "An analytical study of performance towards task-level parallelism on many-core systems using Java API," in *Proc. 6th Int. Conf. Commun. Electron. Syst. (ICCES)*, Jul. 2021, pp. 1255–1259.

[36] Google. Accessed: Sep. 18, 2023. [Online]. Available: https://github.com/google/cluster-data

[37] G. Yao, Q. Ren, X. Li, S. Zhao, and R. Ruiz, "A hybrid fault-tolerant scheduling for deadline-constrained tasks in cloud systems," *IEEE Trans. Services Comput.*, vol. 15, no. 3, pp. 1371–1384, Jun. 2022.

[38] A. Hussain and M. Aleem, "GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures," *Data*, vol. 3, no. 4, p. 38, Sep. 2018.
[39] Z. Liu and S. Cho, "Characterizing machines and workloads on a Google cluster," in *Proc. 41st Int. Conf. Parallel Process. Workshops*, Sep. 2012, pp. 397–403.
[40] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in Google cloud to derive realistic resource utilization models," in *Proc. IEEE 7th Int. Symp. Service-Oriented Syst. Eng.*, Mar. 2013, pp. 49–60.
[41] *Analysis and Lessons From a Publicly Available Google Cluster Trace*. Accessed: Sep. 24, 2023. [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-95.html
[42] C. Reiss, "Towards understanding heterogeneous clouds at scale: Google trace analysis," Ph.D. dissertation, Intel Sci. Technol. Center Cloud Comput., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2012.
[43] C. Liu, C. Liu, Y. Shang, S. Chen, B. Cheng, and J. Chen, "An adaptive prediction approach based on workload pattern discrimination in the cloud," *J. Netw. Comput. Appl.*, vol. 80, pp. 35–44, Feb. 2017.
[44] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production MapReduce cluster," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, May 2010, pp. 94–103.
[45] *Heterogeneous Computing Scheduling Problem*. Accessed: Sep. 18, 2023. [Online]. Available: https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/index.htm
[46] A. A. Motlagh, A. Movaghar, and A. M. Rahmani, "Task scheduling mechanisms in cloud computing: A systematic review," *Int. J. Commun. Syst.*, vol. 33, no. 6, Apr. 2020, Art. no. e4302.
[47] R. H. Riffenburgh, "Chapter summaries," in *Statistics in Medicine* (Second Edition), 2nd ed. New York, NY, USA: Academic Press, 2006, pp. 533–580. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780120887705500678
[48] M. D. Riina, C. Stambaugh, N. Stambaugh, and K. E. Huber, "Chapter 28—Continuous variable analyses: T-test, mannwhitney, wilcoxin rank," in *Translational Radiation Oncology*, A. E. Eltorai, J. A. Bakal, D. W. Kim, and D. E. Wazer, Eds. Cambridge, MA, USA: Academic Press, 2023, ch. 2, pp. 153–163. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323884235000704

ZULFIQAR ALI KHAN received the Bachelor of Science degree (Hons.) in information technology from the Virtual University of Pakistan, and the Master of Science degree in computing from the Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST), Islamabad. He is currently pursuing the Ph.D. degree with Universiti Teknologi PETRONAS (UTP), Malaysia. His research interests include task scheduling, cloud computing, optimization, fog computing, meta-heuristics algorithms, and the Internet of Things (IoT).

IZZATDIN ABDUL AZIZ received the Ph.D. degree in information technology from Deakin University, Australia, working in the domain of hydrocarbon exploration and cloud computing. He is currently an Associate Professor with Universiti Teknologi PETRONAS (UTP), Malaysia. He is also heading the Center for Research in Data Science (CeRDaS), solving complex upstream and downstream Oil and Gas (O&G) industry problems utilizing machine learning and big data analytics. He is also working closely with O&G companies in delivering solutions for complex problems, such as Offshore O&G pipeline corrosion rate prediction, O&G pipeline corrosion detection, rotating machines, and process failure prediction, securing data on clouds, and bridging upstream and downstream oil and gas businesses through data analytics. Additionally, he is also working on fundamental computer science problems, such as algorithm's optimization.

NURUL AIDA BT OSMAN is currently a Lecturer with Universiti Teknologi PETRONAS and a Researcher with the Centre of Research in Data Science (CeRDaS). She served in MIMOS Berhad under the Artificial Intelligence Research Laboratory. Her research interests include machine learning, predictive analytics, recommender systems, ontology development, and sentiment analysis.

SAID NABI received the Ph.D. degree in computer sciences from the Capital University of Science and Technology (CUST), Islamabad. He is currently a Lecturer with the Department of Computer Sciences and Information Technology, Virtual University of Pakistan, Islamabad. His research interests include cloud computing (cloud jobs/applications and resource scheduling, cloud load-balancing, optimization, SLA, and the quality of services (QoS) aware cloud scheduling), machine learning, cloud applications/services development, fog computing, the IoT, big data, and recommender systems. He also served with the I.T Excellence Center, Directorate of Information Technology, Peshawar, and Esided Solutions (a U.S. based company) as a Software Developer and a Team Lead.