IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 13 September 2024, accepted 13 October 2024, date of publication 21 October 2024, date of current version 4 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3484388

RESEARCH ARTICLE

A Neighborhood Inspired Multiverse Scheduler for Energy and Makespan Optimized Task Scheduling for Green Cloud Computing Systems

SHALINI TIWARI AND BEENA B. M., (Senior Member, IEEE)
Department of Computer Science and Engineering, Amrita School of Computing, Amrita Vishwa Vidyapeetham, Bengaluru 560035, India
Corresponding author: Beena B. M. (bm_beena@blr.amrita.edu)

ABSTRACT In the digital era, cloud computing is vital for scalable and efficient infrastructure, but its growing energy consumption raises serious environmental concerns. Green cloud computing strategies, particularly efficient task-scheduling algorithms, are key to addressing this challenge. Task scheduling in cloud computing is NP-hard due to the complexity of managing numerous tasks, resources, and optimization metrics. To address this, we propose a novel task scheduling algorithm named NIMS (Neighborhood Inspired Multiverse Scheduler), designed to optimize two often conflicting metrics: makespan and energy consumption. NIMS improves the performance of the original MVO (Multiverse Optimizer) by incorporating a novel fitness-based neighborhood search strategy during solution updates. This enhancement improves the quality of solutions, particularly when the standard update mechanism of MVO underperforms. By promoting a more effective exploration of the solution space, our approach significantly enhances the algorithm's convergence rate. Further, we performed a comprehensive performance evaluation of the proposed NIMS algorithm against seven advanced algorithms: EMVO, IMOMVO, OPSO, LJFPPSO, TSIGA, FPGWO, and MVO, using the CloudSim toolkit under various test scenarios, leveraging three widely adopted real-world benchmark datasets. Our extensive simulations and experiments exhibit that the proposed NIMS algorithm significantly outperforms the competing algorithms across five key performance metrics: makespan, energy consumption, throughput, load imbalance, and average resource utilization.

INDEX TERMS Green cloud computing, task scheduling, multiverse optimizer (MVO), neighborhood search, local search, metaheuristics.

I. INTRODUCTION

Cloud data centers are the backbone of our current digital infrastructure. Industries widely adopt cloud computing services due to their exceptional flexibility, scalability, and cost-effectiveness. The IaaS (Infrastructure-as-a-Service) model of cloud computing provides various heterogeneous computing, storage, and network resources on a pay-per-use basis to address a variety of computational and data-intensive applications. Cloud service providers physically host the aforementioned cloud services on high-computing servers at their data centers and offer them to cloud users via the Internet.

A recent report from Markets and Markets projects the growth of the global cloud computing market from USD 626.4 billion in 2023 to USD 1266.4 billion by 2028 [1]. However, this growth rate also indicates an increased reliance on data centers, raising concerns about energy consumption and conservation as the demand for data processing and storage continues to grow exponentially.

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

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

To further illustrate, another report claims, ''Currently, the carbon emissions produced by the cloud surpass those of the aviation sector. One data center can consume electricity equivalent to that of 50,000 households'' [2]. Given the expanding market for cloud services and the associated increase in carbon footprint, there is an absolute need for sustainable practices to conserve energy, which is a limited resource, across data centers. In this context, current research trends emphasize the advancement of green cloud computing practices. Green Cloud Computing proposes various strategies to mitigate the cloud industry's environmental impact. The key initiatives involve adopting energy-efficient server architectures and cooling systems, integrating renewable energy sources into the data center's operations, and implementing energy-aware algorithms for application scheduling and resource allocation.

The IaaS model of cloud computing uses virtualization technology to improve the utilization of underlying physical resources by distributing the CPUs (Central Processing Unit) and other resources among multiple VMs (virtual machines). Many cloud service providers then provision these VMs as a service to execute user-submitted tasks. Although assigning multiple virtual machines over physical servers improves physical resource utilization, the inefficient use of instantiated VMs remains a significant concern. If not utilized optimally, these VMs can negatively affect critical performance metrics, such as execution time, cost, and energy consumption, ultimately slowing down overall system performance. Hence, it is essential to devise an effective task-scheduling algorithm to enhance the system's performance.

In cloud computing, the problem of scheduling tasks involves the optimal allocation and distribution of a set of user tasks across a set of available virtual machines. However, optimizing several performance criteria, such as the finish time of the overall schedule (makespan), the execution cost, system throughput, load distribution, and energy consumption, constrains the algorithm's efficiency [30]. Moreover, efficiently allocating tasks to available VMs while simultaneously optimizing these performance parameters is considered an NP-hard problem [12], [13], [16], mainly due to the vast solution space involved.

Numerous heuristics and rule-based algorithms were proposed in the past, but they often overlook the better alternative solutions due to their adherence to predefined rules [3], [4], [5], [6]. Also, these approaches are better suited to small-scale problems where the number of tasks and VMs is smaller. However, their efficacy degrades with the expansion of problem size and the number of optimization objectives [24], [31]. Compared to conventional algorithms, metaheuristic algorithms explore a vast solution space simultaneously while optimizing selected performance metrics to reach solutions close to optimality in polynomial time rather than exponential time. These algorithms are proven effective in solving a diverse range of complex and real-world optimization problems [30]. However, their solution update mechanism frequently experiences an imbalance between exploring and exploiting the search space, potentially trapping the algorithm in the local solution.

The multiverse optimizer (MVO) is one such metaheuristic algorithm that has proven its efficacy over other metaheuristic algorithms in solving many real-world optimization problems [42], such as optimal band selection for hyperspectral image data [7], [8], text document clustering [9], energy estimation [10], and task scheduling problems [12], [13], [14], [15]. Thus, in this study we have selected a multiverse optimizer to solve the problem of task scheduling optimization in cloud computing systems. Also, from the literature review, it is observed that MVO has not been previously applied to optimize energy consumption in task scheduling problems. So, we have combined the two critical scheduling parameters, makespan and energy consumption, as our optimization objective.

However, the following issues are possible in the direct application of MVO for the task scheduling optimization problem:

1) The original MVO algorithm is designed for solving continuous optimization problems, while the task scheduling problem belongs to combinatorial optimization.
2) The original MVO's solution update mechanism primarily relies on random changes to existing solutions and does not incorporate problem-specific heuristic information about the optimization parameters during updates. This limitation may hinder its ability to effectively explore optimal solutions within the solution space.
3) MVO suffers from the problem of premature convergence in complex and high-dimensional problem spaces.

To address these challenges in this context, we incorporated a novel fitness-based neighborhood search strategy with the standard update mechanism of MVO to achieve an optimal balance between exploring new solutions and exploiting the ones already discovered. This tailored search technique is specifically designed to improve the solution quality as the iteration progresses. Additionally, it promotes quicker convergence and prevents the algorithm from becoming trapped in local optima, leading to more effective optimization outcomes.

The major contributions of our proposed research are discussed below:

1) A dual-objective fitness function is modeled to address task scheduling problem in a heterogeneous cloud environment. The weighted sum approach is used to combine both the major objectives, makespan, and overall energy consumption in a fitness function.
2) A novel fitness-based neighborhood search approach has been designed to integrate with the standard solution update mechanism of the Multiverse Optimizer, enhancing its search capabilities and speeding up convergence. This new solution update mechanism considers two key metrics—task finish time on VMs

TABLE 1. Description of metaheuristic algorithms.

Abbreviation	Description
MVO	Multiverse Optimizer
EMVO	Enhanced Multiverse Optimizer
IMOMVO	Improved multi-objective MVO
LJFPPSO	Largest Job to Fastest Processor based Particle Swarm Optimization
OPSO	Opposition Learning Based PSO
HHO	Harris Hawks optimization
TSIGA	Task Scheduling using an improved Genetic Algorithm
FPGWO	Flower Pollination and Grey Wolf Optimization
CSO	Chameleon Search Optimization
SSA	Salp Swarm Algorithm
MRFO	Manta Ray Foraging Optimization
GWO	Grey Wolf Optimization
WOA	Whale Optimization Algorithm

and the overall increase in energy consumption when replacing a VM within a solution. By incorporating these factors, the approach improves solution quality and enables smarter task distribution across available virtual machines, effectively reducing makespan and minimizing overall energy consumption.

3) An extensive experimental simulation is conducted using the CloudSim toolkit to compare and analyze the performance of the proposed NIMS scheduler with seven other advanced metaheuristic-based task scheduling algorithms, namely EMVO (Enhanced MVO) [13], IMOMVO (Improved multi-objective MVO) [14], OPSO (Opposition Learning Based Particle Swarm Optimizer) [16], LJFPPSO (Largest Job to Fastest Processor-based PSO) [17], TSIGA (Task Scheduling using an Improved Genetic Algorithm) [35], FPGWO (Flower Pollination and Grey Wolf Optimization) [36], and MVO [41]. The results are validated using publicly available and widely used benchmark datasets using five performance metrics: makespan, energy consumption, throughput, degree of imbalance, and average resource utilization.
4) At last, the statistical validation is done using a non-parametric Kruskal-Wallis test on the obtained results. This test assesses whether there are statistically significant differences between the medians of the results from different algorithms, providing a robust comparison of their performance across various metrics.

The organization of the paper in the following sections is as follows: In Section II, we provide an extensive review of existing literature concerning the task scheduling problem in cloud computing. We particularly emphasize the application of diverse metaheuristic algorithms to tackle this challenge. Section III provides a detailed description of the proposed scheduling framework and performance metrics. In Section IV, we state the problem statement for task scheduling in a cloud environment and mathematically formulate the proposed fitness function. We also discuss the standard MVO algorithm and the proposed NIMS algorithm in the same section. The details of the simulation setup involving architectural configurations and workload description are provided in Section V. Further, the experimental findings and observations are provided in Section VI, offering insights into the efficacy and comparative performance of our proposed algorithm against selected state-of-the-art techniques. In Section VII, the statistical analysis of all the comparative algorithms is done using the Kruskal-Wallis test. At last, Section VIII provides a summary of the work accomplished, discusses its limitations, and highlights potential directions for future research.

II. RELATED WORK

In the current section, we reviewed and compared many recently developed algorithms for task scheduling optimization in a cloud computing environment. We have mainly considered the optimization algorithms based on metaheuristic techniques. However, there are many other algorithms also developed to address this problem in [39], [40]. Several metaheuristic-based algorithms were proposed in the past to address the problem of task scheduling optimization in a cloud computing environment. In [11], a detailed analysis of these algorithms was presented with respect to their types and the considered optimization objectives. In the current study, the surveyed papers are divided into two categories: *1) Single Objective Optimization Based* and *2) Multi Objective Optimization Based*. Table 1 provides a description of the short name of these algorithms. Tables 2 and 3 provide a concise overview of these algorithms.

A. SINGLE OBJECTIVE OPTIMIZATION

In [13], Shukri et al. have proposed an enhanced version of the multiverse optimizer to schedule a set of independent tasks in cloud computing environments. In each iteration, a few good-quality solutions are saved and fed back to the optimizer after a few iterations to improve the optimization process. In [15], Abualigah and Alkhrabsheh utilized a hybridization technique to improve the optimization process of MVO. They have used the crossover and mutation techniques of GA (Genetic Algorithm) to amplify the search process of standard MVO. Later, they used this hybrid algorithm, named MVO-GA, to generate an optimized task schedule with a reduced makespan (referred to as task transfer time in the paper). However, they did not validate the efficacy of their approach with any recent baseline algorithms. Also, the length of the tasks considered for the evaluation is very small.

Further, Agarwal and Srivastava in [16] and Alsaidy et al. in [17] have proposed two distinct versions of PSO (Particle Swarm Optimization) to address the task scheduling problem. In [16], opposition-based learning is applied to accelerate the convergence speed while mitigating the issue of premature convergence of the traditional PSO algorithm. Using the CloudSim simulator, they validated the algorithm against other baseline algorithms in terms of makespan, degree of imbalance, performance improvement ratio, and energy consumption. Whereas, in [17], the authors have combined two well-known heuristic algorithms, namely LJFP (Largest

TABLE 2. Summary of single objective optimization algorithms.

Meta-heuristic	Improvisation Technique	Optimization Objective	Performance Metrics	Workload DataSet
Multiverse Optimization [13, 15]	Elitism Technique	Makespan	Makespan Throughput Avg. Resource Utilization	Google Cluster Inspired (GoCJ) Real Workload Traces
	Hybridization with Genetic Algorithm	Makespan	Makespan	Synthetic Dataset
Particle Swarm Optimization [16, 17, 20]	Opposition Learning Based Updation	Makespan	Makespan Degree of imbalance Energy consumption	Synthetic Dataset
	Heuristic Initialization using LJFP and MCT	Makespan	Makespan Execution Time Degree of Imbalance Energy Consumption	Synthetic Dataset
	Probabilistic Neighborhood Search	Makespan	Makespan Degree of Imbalance Energy Consumption	Synthetic Dataset
Harris Hawk Optimization [31]	Hybridization with Simulated Annealing	Makespan	Makespan Convergence Rate	HPC2N, NASA-iPSC and Synthetic Dataset
Discrete Prey Optimization [32]	NA	Makespan	Makespan	Synthetic Dataset
Manta-Ray Foraging [34]	Hybridization with Salp Swarm Optimization	Makespan	Makespan Throughput	HPC2N and NASA-iPSC Workloads
Henry gas solubility Optimization [37]	Hybridization with Whale Optimization	Makespan	Makespan	Synthetic, HPC2N and NASA-iPSC Workloads

Job to Fastest Processor) and MCT (Minimum Completion Time), in solution initialization to improve the solution quality and convergence speed of the algorithm. They compared their algorithms on four objectives: makespan, complete execution time, degree of imbalance, and energy consumption. However, both algorithms have considered synthetically generated tasks for comparison and have not considered energy consumption as an optimization objective.

Shamaa et al. [20] suggested a novel approach for scheduling tasks in cloud computing. The algorithm combines PSO with a probabilistic neighborhood search strategy to make particle selection and position update mechanisms better. They evaluated the algorithm's efficacy by optimizing three performance measures: makespan, degree of imbalance, and energy consumption. In their future work, they suggested combining the proposed neighborhood search with other metaheuristic algorithms for further improvement.

Further, the authors in [31] introduced a modified version of the HHO (Harris Hawks optimization) algorithm incorporating SA (Simulated Annealing) for scheduling jobs in the cloud computing environment. By integrating simulated annealing as a local search algorithm, the standard HHO algorithm achieves better convergence rates and higher-quality solutions. However, this work focuses solely on optimizing makespan and does not take energy consumption into account. Ibrahim Attiya et al. [34] have developed an innovative hybrid swarm intelligence technique integrating a modified MRFO (Manta Ray Foraging Optimization) and the SSA (Salp Swarm Algorithm) to address IoT task scheduling issues in cloud computing. This proposed method, named MRFOSSA, relies on SSA to refine the local search ability of MRFO, thereby facilitating faster convergence to the global solution. The other work [37] proposed by the same author is designed for optimum task scheduling utilizing the Henry gas solubility optimization. In this work, the intermediate solutions were improvised by integrating WOA's (Whale Optimization) local search mechanism, whereas comprehensive opposition-based learning is also incorporated to enhance the quality of the worst solutions by finding their opposite solutions and selecting the optimal one. However, the algorithm lacks to prove its validity over advanced algorithms developed to handle similar problems.

B. MULTI OBJECTIVE OPTIMIZATION

In [12], Nekooei-Joghdani and Safi-Esfahani suggest a new hybrid metaheuristic algorithm that can help solve the problem of how to best map a set of parallel tasks to a given set of virtual machines. This work primarily focuses on two categories. Firstly, they have integrated the standard MVO algorithm with the Gabor Filter, Opposition-Based Learning, and Multi-Tracker Optimization to enhance the convergence speed and prevent the algorithm from getting trapped in a local optimum solution. Further, the algorithm is applied to solve the task-to-VM mapping problem to optimize four performance parameters: response time, execution time, throughput, and SLA (Service Level Agreement) violation. They evaluated their algorithm using real workload traces from NASA-iPSC, comparing it to some baseline algorithms. However, their future work suggests including other performance criteria, such as cost of the computation, energy consumption, and effect of migration,

TABLE 3. Summary of multi objective optimization algorithms.

Meta-heuristic	Improvisation Technique	Optimization Objective	Performance Metrics	Workload DataSet
Multiverse Optimization [12, 14]	Opposition based learning, Gabor filter and Multi-tracker Optimization	Not mentioned	Execution Time Response Time Throughput SLA Violation	NASA-iPSC Workloads
	Novel Average Positioning Methods	Not mentioned	Makespan Throughput	Synthetic Dataset
Particle Swarm Optimization [18, 19, 21]	Hybridization with Chemical Reaction Optimizer	Execution Cost Makespan Energy Consumption	Execution Cost Makespan Energy Consumption	Synthetic Dataset
	TOPSIS based Task Prioritization	Makespan Transmission Time Cost and Resource Utilization	Makespan Transmission Time Cost and Resource Utilization	Synthetic Dataset
	Orthogonal Matrix based initialization Rerandomization and Adaptive inertia weight calculation	Not mentioned	Cost and Time of Execution	Synthetic Dataset
Wild horse Optimization [22]	Levy Flight based search	Resource Utilization Makespan	Resource Utilization Makespan	Synthetic Dataset
Chameleon Search Optimization [23]	Hybridization with Remora search Optimization	Makespan Load Cost of Execution	Makespan Cost of Execution Degree of Imbalance Resource Utilization Energy Consumption	Synthetic Dataset
Grey Wolf Optimization [24, 35]	NA	Makespan Throughput Degree of Imbalance Resource Utilization	Makespan Throughput Degree of Imbalance Resource Utilization	Synthetic HCSP and GoCJ Dataset
	Hybridization with Genetic Operators	Makespan Energy Consumption Cost of Schedule	Makespan Energy Consumption Cost of Schedule	Synthetic Dataset HPC2N, NASA-iPSC Workloads
Non-dominated sorting Genetic Algorithm (NSGA-II) [25]	NA	Makespan Energy Consumption Cost of Schedule	Makespan Energy Consumption Cost of Schedule	Synthetic and HCSP Dataset
Whale optimization [26, 27, 38]	NA	Not mentioned	Makespan Cost	Synthetic Dataset
	Adaptive Population Size and Convergence Factor	Execution Time Load Cost	Execution Time Load Cost	Synthetic Dataset
	Levy Flight based search	Task Completion Time Resource Utilization Energy Consumption	Task Completion Time Resource Utilization Energy Consumption	Synthetic Dataset HPC2N Workload
Genetic Algorithm [28, 29]	Energy Conscious Scheduling Heuristic	Makespan Energy	Makespan Energy	Synthetic Dataset
	Hybridization with Ant Colony Optimization	Makespan Convergence Time	Makespan Convergence Time Cost Response Time	Synthetic Dataset
Spider Monkey Optimization [30]	Hybridization with Ant Colony Optimization	Makespan Execution Cost Energy Consumption	Makespan Execution Cost Energy Consumption Resource Utilization Degree of Imbalance	HPC2N workload Synthetic Dataset
Cat Swarm Optimization [33]	Priority Based Scheduling	Makespan Energy Consumption Power Cost Migration Time	Makespan Energy Consumption Power Cost Migration Time	HPC2N NASA-iPSC workload
Flower Pollination [36]	Hybridization with Crossover and Grey Wolf Optimizer	Makespan Degree of Imbalance Resource Utilization Throughput	Makespan Degree of Imbalance Resource Utilization Throughput	Synthetic HPC2N GoCJ workload
Heuristic Algorithm [39]	TOPSIS based Selection	Cost of Communication Execution cost Turnaround time Grid utilization	Cost of Communication Execution cost Turnaround time Grid utilization	Synthetic and Scientific Workflows

in order to address the task scheduling problem. In addition, they have proposed conducting tests and measuring the performance of their algorithm using diverse datasets in the future.

Similarly, Otair et al. [14] developed a novel average positioning-based improved MVO algorithm, IMOMVO, to address the issue of task scheduling in cloud computing. The average positioning method is used to improve the updating mechanism of standard MVO. The results show their efficacy over standard MVO and PSO for a large dataset. However, comparison with more competitive algorithms is lacking in this paper.

Dubey and Sharma [18] suggested a hybrid optimization algorithm for task scheduling called Chemical Reaction Partial Swarm Optimization. They used this algorithm to allocate multiple distinct tasks to the available virtual machines. They combined the best features of both optimizers to improve factors like cost, energy, and makespan. Similarly, TOPSIS-PSO is used in [19] to improve execution time, transmission time, and schedule cost. They applied the TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) algorithm to measure the relative closeness of tasks with respect to the defined scheduling criteria., which is used by PSO for further optimization. In [21], a fuzzy-based PSO algorithm is presented for solving task scheduling problems in a cloud computing environment to optimize the time and cost of the schedule. This work assumes uncertainty in a task's expected execution time and uses a fuzzy system to determine the task's uncertain execution time on a virtual machine. Further, several techniques, such as orthogonal matrix-based initialization, re-randomization, and adaptive inertia weight calculation, are used to optimize the convergence speed and avoid premature convergence of the algorithm.

Sarvanan et al. [22] have solved the problem of task scheduling in cloud computing using another metaheuristic algorithm based on Wild-Horse Optimization. During each iteration, they utilize the levy flight technique to generate improved solutions, thereby preventing the algorithm from becoming trapped in a non-optimal solution. In [23], the authors have presented a hybrid optimization algorithm combining the advantages of Chameleon Search and Remora Search Optimization to solve a similar problem. The simulation results evidently validated that the suggested CRSOA approach minimizes the execution time, cost, and load imbalance between the available VMs when compared to other competitive metaheuristic-based task scheduling algorithms. However, in both papers, energy consumption is not considered as an optimization parameter. Additionally, the work suggests testing the algorithm's performance on real cloud task traces as a future focus. Also, D. Alsadie, in his works [24], [25], has tried to optimize task scheduling using GWO (Grey Wolf Optimizer) and NSGA-II (Non-dominated Sorting Genetic Algorithm), respectively. These works have come up with multi-objective frameworks to optimize various performance parameters. However, the work suggests considering the complexities of future scheduling environments.

Three different variations of whale optimization-based task scheduling algorithms have been proposed in [26], [27] and [38]. In [26], the objective is to minimize the overall makespan and execution cost of the schedule. However, this work did not consider the energy consumption in the evaluation criteria. Also, the evaluation only considered small synthetically generated tasks and did not provide any comparison with other related studies. Similarly in [27], a multi-objective task scheduling optimization algorithm is proposed to optimize three major objectives: execution time, resource load, and resource cost. The authors have developed a new technique to dynamically change the population size during generations, and a novel convergence factor is also integrated to balance between exploration and exploitation for improving the search process. They have conducted various experiments, including small and large workloads, to test the efficacy of their algorithm. In [38], the authors have developed and enhanced version of whale optimization algorithm to tackle multi-objective scheduling. The Lévy fight movement is used to expand the search space of the WOA, enabling it to investigate a broader spectrum of solutions. The optimization criteria involved energy consumption, resource utilization, and task completion time. However, the efficacy is tested only over the basic meta-heuristic algorithm.

Further, two different works in [28], [29] have developed multi-objective optimization algorithms to deal with the development of efficient task scheduling in cloud computing systems. Pirozmand et al. in [28] have designed a two-step approach where, in the first step, their algorithm prioritizes the arrived tasks, and then an energy-aware task assignment step called GAECS (Genetic Algorithm based Energy Conscious Scheduling) is applied to efficiently assign the tasks on the best processor with lesser voltage input. The algorithm focuses on achieving an optimal trade-off between time and energy usage in the task assignment process for processors. Analysis with MATLAB demonstrated that the GAECS algorithm provides superior accuracy compared to other algorithms. Whereas, in [29], the authors have divided large workloads into smaller groups and applied Genetic Algorithm to optimize the allocation of each subgroup to a given set of virtual machines. In the process of scheduling, they further detected the overloaded VMs after each sub-group allocation to effectively distribute the tasks amongst VMs. However, both of these approaches have not evaluated their algorithms efficacy on any real-world workload traces.

Amer et al. in [30] have developed a multi-objective scheduling algorithm by combining the best features of two metaheuristic algorithms, spider monkey optimizer and ant colony optimizer. The major focus of this work is to optimize four important scheduling metrics: schedule length, execution cost, energy consumption, and resource utilization. In papers [35] and [36], two hybrid task scheduling algorithms are proposed based on Grey Wolf Optimization (GWO) and Flower Pollination (FP) algorithms, respectively. In [35], the authors have utilized the crossover and mutation operators of the Genetic Algorithm to improve

the optimization mechanism of the original GWO while minimizing the makespan, energy consumption, and cost of the schedule. However, the algorithm's performance is only not tested against any recently developed advanced algorithms dealing with task scheduling. Whereas, in [36], an improved flower pollination algorithm is developed to deal with task scheduling to optimize throughput, makespan, degree of imbalance, and resource utilization. They have used crossover operations to improve the solution quality in flower pollination, and also GWO operators are used to improve the worst solution. The efficacy of the algorithm is proven over three large datasets.

A multi-objective task scheduling framework is presented in [39]. The article introduces a task scheduling algorithm utilizing multi-objective optimization to enhance grid utilization while simultaneously reducing turnaround time, execution costs, and data communication expenses in a heterogeneous multi-grid environment. Integrated into the GridSim architecture, the framework includes five schedulers: a Greedy scheduler optimizing multiple factors, communication cost-focused, execution cost-focused, no fragmentation, and a Random scheduler. Each scheduler targets specific objectives to enhance task distribution across computing resources. By leveraging heuristic approaches, the framework offers practical and computationally efficient solutions for complex scheduling challenges, efficiently handling conflicting objectives. In other work by Stewart et al. [40], a bi-objective mixed integer programming (MIP) formulations have been proposed to address both schedule efficiency and energy usage, with Pareto-efficient solutions offering valuable trade-offs. Additionally, methods like the dichotomic and box techniques have been employed to analyze non-dominated solutions, highlighting their strengths and limitations in computational efficiency.

At last, the following observations are made from the above literature:

1) Energy consumption is not considered a primary optimization objective in many of these works.
2) Most studies focus on using either a single metaheuristic algorithm or a combination of algorithms to improve solution quality and convergence. These approaches often rely on stochastic processes and random variations to explore the solution space, which can be effective for some types of problems. However, this randomness may not be ideal for discrete and highly structured problems, such as task scheduling, where it can lead to inefficient exploration and suboptimal results. Therefore, a problem-specific search strategy is necessary to enhance the optimization process and achieve better performance for such problems.
3) The testing and validation of the proposed algorithms on real-world benchmark datasets and comparison with recently developed advanced algorithms is lacking in many of the works.

III. PROPOSED SCHEDULING FRAMEWORK AND PERFORMANCE METRICS

There are two subsections in this section. The first subsection outlines the proposed scheduling framework and related models for the problem. The second subsection presents a mathematical representation of the analyzed performance metrics.

A. PROPOSED SCHEDULING FRAMEWORK AND MODEL DESCRIPTION

This article proposes a new approach to optimize task scheduling using the NIMS algorithm in a heterogeneous cloud computing environment. The proposed scheduling framework and the resource model are displayed in Fig. 1. Various cloud service providers, such as Amazon AWS, Microsoft Azure, Google Cloud, etc., offer a variety of VM instances running on physical servers called hosts through a virtual machine monitor (VMM) or hypervisor. The VMM logically partitions the host's underlying physical resources and allocates them separately to individual VMs. The Task Queue temporarily stores the tasks that users submit for processing. These tasks are further retrieved by the Task Manager. The task manager then invokes the task scheduling model, which employs the proposed NIMS algorithm to begin the optimization process and provides an optimized schedule. This schedule gives a suitable mapping of selected tasks over optimally chosen VMs.

The related task and resource models are as follows:

1) TASK MODEL

In a cloud environment, users can submit a variety of tasks, such as batch or interactive jobs. The batch jobs are mainly independent in nature, whereas the interactive jobs are dependent on various events. These tasks' requirements vary depending on their length and required resources. In this work, we consider a set of 'n' independent tasks represented by $T = \{T(1), T(2), \ldots T(n)\}$. Each task $T(i)$ submitted by a user is distinguished by a unique identification number $T_{id}(i)$ and its length $T_{length}(i)$, measured in millions of instructions (MI), as shown below.

$$T(i) = [T_{id}(i), T_{length}(i)]$$

2) VM MODEL

As discussed in previous sections, our major focus in this work is to efficiently schedule a given set of independent tasks over available virtual machines. To mathematically model the resource, we consider a set of 'm' virtual machines represented by $VM = \{VM(1), VM(2), \ldots VM(m)\}$. Each of these VMs has a specific configuration in terms of number of cores or CPUs, processing speed per core in million instructions per second (MIPS), and RAM. The features of $VM(j)$ are defined here.

$$VM(j) = [VM_{id}(j), VM_{cpu}(j), VM_{mips}(j), VM_{ram}(j)]$$

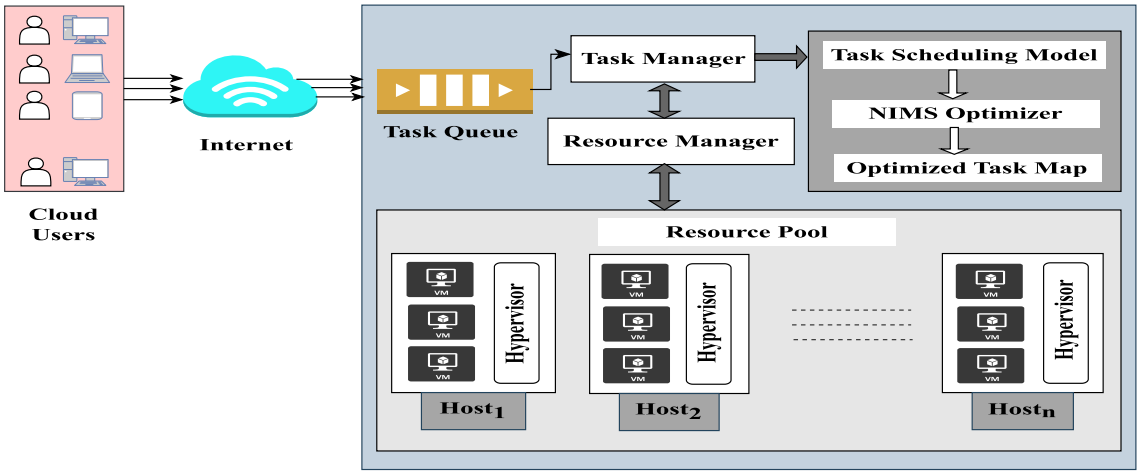

FIGURE 1. Task-scheduling-framework.

3) TASK EXECUTION TIME MODEL

Similar to the works in [17], [30], [31], we formulated our scheduling problem based on the "Expected Time to Compute" (ETC) model. The ETC model computes the expected execution time for each task $T(i)$ on each machine $VM(j)$ using Eq. 1. The computed values are represented in the form of a matrix of size "n x m," where 'n' is the number of tasks and 'm' is the count of VMs. Each entry $ETC_{i,j}$ represents the expected computation time of a task $T(i)$ on virtual machine $VM(j)$, as shown in Eq. 2.

$$ETC_{i,j} = \frac{T_{length}(i)}{VM_{cpu}(j) \times VM_{mips}(j)} \tag{1}$$

$$ETC_{i,j} = \begin{bmatrix} ETC_{1,1} & ETC_{1,2} & \cdots & ETC_{1,m} \\ ETC_{2,1} & ETC_{2,2} & \cdots & ETC_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ ETC_{n,1} & ETC_{n,2} & \cdots & ETC_{n,m} \end{bmatrix} \tag{2}$$

B. PERFORMANCE METRICS

A task scheduling algorithm's primary goal is to generate an optimal or near-optimal schedule that maps the tasks onto a set of virtual machines by optimizing various performance metrics. The cloud user and the service provider divide these performance metrics into two categories based on their respective requirements [36], [48]. The cloud user majorly focuses on reducing makespan and cost of execution, whereas the service provider considers maximizing throughput and average resource utilization while minimizing the degree of imbalance and energy consumption as their optimization objectives. This subsection discusses the mathematical formulation for the performance measures considered in our work.

1) MAKESPAN (MS)

The makespan is defined as the maximum duration required by a group of virtual machines to complete a set of assigned tasks. It is one of the most widely used metrics to evaluate the performance of any scheduling algorithm in cloud computing. Its lower value indicates a better scheduling solution. To calculate the makespan of a given schedule, we first calculate the finish time VM_{finish} for all VMs by Eq. 3 and then determine the VM with maximum finish time to find the makespan using Eq. 4.

$$VM_{finish}(j) = \sum_{i=0}^{n-1} ETC_{i,j} \times A_{i,j}, \ \forall VM(j) \tag{3}$$

where 'n' is the number of tasks, $ETC_{i,j}$ is the execution time of ith task on j^{th} VM calculated using Eq. 1 and

$$A_{i,j} = \begin{cases} 0, & \text{if task } T(i) \text{ is not allocated to } VM(j) \\ 1, & \text{if task } T(i) \text{ is allocated to } VM(j) \end{cases}$$

$$MS = \max \sum_{j=0}^{m-1} VM_{finish}(j) \tag{4}$$

2) ENERGY CONSUMPTION (EC)

The total energy consumed for a schedule is expressed as the energy used by all the VMs during overall makespan length. This energy consumption is divided into two VM modes: *the execution mode* and *the idle mode*. Many studies state that the energy consumption in idle mode is almost 60-70% of the energy consumption in execution mode, whereas the energy consumption in execution mode depends upon the execution speed of the virtual machine. Accordingly, the overall energy consumption for a schedule is modeled as Eq. 5.

$$EC = \sum_{j=0}^{m-1} (VM_{exec}(j) \times \alpha_j + VM_{idle}(j) \times \beta_j) \tag{5}$$

where α_j and β_j are j^{th} VM's power consumption in execution mode and idle mode, respectively. Each VM's time spent in execution mode and idle mode is calculated by Eqs. 6 and 7, respectively.

$$VM_{exec}(j) = VM_{finish}(j), \quad \forall VM(j) \tag{6}$$

$$VM_{idle}(j) = MS - VM_{exec}(j), \quad \forall VM(j) \tag{7}$$

Finally, the lower value of total energy consumption is a desired criteria of our optimization problem.

3) THROUGHPUT (TH)

System throughput signifies the number of tasks successfully executed per unit of time. It reflects the efficiency of the scheduling algorithm in maximizing resource utilization and minimizing idle time, with higher throughput indicating that more tasks are processed in a given time frame. Maximizing throughput is a key objective in optimizing scheduling performance, especially in multi-tasking environments. It is measured using Eq. 8.

$$Th = \frac{Total\ number\ of\ completed\ tasks}{Makespan} \tag{8}$$

4) DEGREE OF IMBALANCE (DOI)

In task scheduling, the degree of imbalance measures how unevenly tasks are distributed across available resources. It quantifies the deviation from an ideal balanced state, where each resource handles an equal share of tasks. A higher degree of imbalance indicates that some resources are overloaded while others are underutilized, leading to inefficiencies and increased execution time. Minimizing the degree of imbalance is crucial for optimizing performance, ensuring efficient use of resources, and reducing the overall makespan of scheduled tasks. It can be computed using Eq. 9.

$$DoI = \frac{MS - \min(VM_{finish})}{\overline{VM_{finish}}} \tag{9}$$

where $\min(VM_{finish})$ represents the smallest finish time among all VMs and $\overline{VM_{finish}}$ is the average finish time of all VMs.

5) AVERAGE RESOURCE UTILIZATION RATE (ARUR)

Another essential metric that concerns the cloud service provider is maximizing the average resource utilization in order to increase profitability from a limited number of resources. Its higher value indicates the system's ability to make better use of its resources. The average resource utilization of all VMs involved in a schedule can be calculated as Eq. 10.

$$ARUR = \frac{\sum_{j=0}^{m-1} Util(VM(j))}{Number\ of\ VMs} \tag{10}$$

where $Util(VM(j))$ represents the % of time $VM(j)$ is actively busy in executing tasks during the overall makespan. It is calculated using Eq. 11.

$$Util(VM(j)) = \frac{VM_{finish}(j)}{MS} \tag{11}$$

In the above equation $VM_{finish}(j)$ denotes the total time spent by the VM in execution mode.

IV. PROBLEM DESCRIPTION AND PROPOSED ALGORITHMS

This section has three parts: the problem definition and proposed fitness function; the standard MVO algorithm; and the proposed NIMS algorithm.

A. PROBLEM DEFINITION AND FITNESS FUNCTION

As previously discussed, the task scheduling problem in a cloud computing environment involves the optimal mapping of a set of tasks across a set of VMs. In this work, we formulate the task scheduling problem as a dual-objective optimization problem to optimize the makespan and energy consumption simultaneously for the generated schedule. Thus, in Eq. 12, we define the mathematical formulation of the proposed dual objective fitness function based on the makespan and energy consumption metrics defined in Section III. Both of these metrics are supposed to be minimized for an optimal schedule. Thus, we combine makespan and energy consumption into an integrated fitness function by the weighted sum technique, which is frequently utilized in the literature and is versatile enough to accommodate various numerical objectives because of its simplicity.

$$\min(Fitness) = w_1 \times MS + w_2 \times EC \tag{12}$$

where w_1 and w_2 denote the weighting factors for each metric. These weights decide the relative importance of each criterion in the overall optimization process. The summation of w_1 and w_2 should be 1. In this work, we consider an equal value of 0.5 for both factors to ensure fairness and avoid any bias towards a specific objective. Further, we have used the normalized values of each metric to account for potential differences in their scale. The following constraints are designed for the scheduling process [16], [24], [25]:

1) The tasks considered are independent in nature and can be executed simultaneously.
2) Each task can be assigned to a single VM and is not allowed to migrate in between its execution.
3) Any VM with sufficient computing resources can handle a task.
4) The virtual machines exhibit heterogeneity in terms of their processing capacity and power consumption.
5) We assume a first-in-first-out (FIFO) execution order for tasks assigned to the same virtual machine.

B. STANDARD MVO ALGORITHM

Originally proposed by Mirjalili et al. [41], the multiverse optimizer (MVO) is a recently developed nature-inspired algorithm that is based on the idea of multiverse physics. According to this algorithm, when the two universes collide with each other, they tend to create a new universe. Three cosmological concepts inspire the main concepts of MVO: white holes, black holes, and worm holes. The white holes are considered a main component of creating a new universe. The black holes are the areas of the universe that attract everything due to their high gravitational force, whereas the wormhole is a tunnel through which objects can travel

between two universes. These possible solutions are called universes and represented as a matrix U, given in Eq. 13. The 'n' rows signify the number of universes, while the 'd' columns signify the number of objects within each universe. To facilitate object exchanges, each iteration creates white holes, black holes, and wormholes between these universes, equivalent to exchanging the variables of two solutions. The exchange process switches between two major phases that are important for any optimization algorithm: **(a) the exploration phase** and **(b) the exploration phase**. The detailed steps involved in the original multiverse optimization are presented in Algorithm 1.

$$U = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^d \\ x_2^1 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ x_n^1 & x_n^2 & \cdots & x_n^d \end{bmatrix} \tag{13}$$

1) EXPLORATION PHASE

During this phase, universes with better fitness tend to have white holes, whereas universes with lower fitness have a greater probability of having black holes, so they attract objects from white holes. This property allows low-quality solutions to attract and retrieve some of the characteristics of high-quality solutions. Lines 12–15 in Algorithm 1 explain the procedure followed for the exploration phase. To start the optimization process, the optimizer first sorts the universes in **SU** according to their normalized fitness function from best to worst, based on the maximization or minimization problem. After that, in line 10, each universe is first considered a black hole to allow updates in its objects. Further, a random number $rand_1$ is generated in the range [0,1). If $rand_1$ is less than the normalized fitness value of the current universe, then a better universe is randomly selected as a white hole from the sorted universes using Roulette Wheel Selection and shares its object with the current universe. The mathematical model representing this exchange between the white hole and black hole universes is displayed in Eq. 14.

$$x_i^j = \begin{cases} x_k^j, & rand_1 < NF_i \\ x_i^j, & rand_1 \geq NF_i \end{cases} \tag{14}$$

where x_i^j is the jth object of the ith universe with black hole, NF_i is the normalized fitness of the ith universe, and $rand_1$ is a randomly generated number in range [0,1). x_k^j is the jth object of the kth universe selected by the roulette wheel (white hole).

2) EXPLOITATION PHASE

The exploitation phase of MVO optimizer is shown in lines 16–22 of Algorithm 1. During this phase, the optimizer assumes the presence of a wormhole between the best universe and other universes to allow exploitation of the current universe towards the best universe. The optimizer creates these wormholes based on a selection probability of 0.5 as described in [38]. The exchange process is defined in Eq. 15, as shown at the bottom of the next page, where $BU(j)$ is the jth object of the Best Universe found so far and TDR is the Traveling Distance Rate coefficient to define the distance rate (variation) at which an object can be transferred by a wormhole around the Best Universe found so far. WEP is another coefficient known as Wormhole Existence Probability to define the probability of exchange between the current universe U_i and the current BU, and $rand_2$, $rand_3$, and $rand_4$ are random numbers in the range [0,1).

The value for WEP is linearly increased, while the value for TDR is linearly decreased using Eqs. 16 and 17, respectively, over iterations to emphasize exploitation around the best universe.

$$WEP = WEP_{min} + curr_{iter} \times \left(\frac{WEP_{max} - WEP_{min}}{max_{iter}}\right) \tag{16}$$

$$TDR = 1 - \left(\frac{{curr_{iter}}^{1/p}}{{max_{iter}}^{1/p}}\right) \tag{17}$$

Here, p defines the exploitation accuracy over iterations which is chosen as '6' according to [41].

C. PROPOSED NIMS ALGORITHM

This section explains the proposed Neighborhood Inspired MVO Scheduler (NIMS) algorithm in detail. This algorithm extends the basic update mechanism of the MVO algorithm explained in the above section to solve the optimization problem of task scheduling in an efficient way. The major improvements done in the proposed NIMS algorithms are as follows:

1) The proposed NIMS algorithm applies a novel fitness-based neighborhood search strategy to update an intermediate solution in case the original update of MVO does not perform well. This additional mechanism strengthens the exploitation around the current solution by utilizing the two important heuristic information: the finish time of a task on a VM and the increase in energy consumption if a task will be assigned to that VM. By including these factors, the algorithm tries to select better VMs to improve the solution quality.
2) Further, a roulette wheel selection is applied in the VM selection process to avoid the algorithm always selecting the fittest VM in the neighborhood. This addition preserves the diversity of the solutions and avoids the algorithm becoming trapped in local solutions.
3) At last, the neighborhood selection probability NSP is defined to control the search process during iterations. The value of NSP linearly decreases over iterations to avoid major changes in the solution gradually.

The steps involved in NIMS are explained in algorithms 2 and 3. A basic flowchart of the proposed algorithm is presented in Fig. 2. The following subsection describes the steps involved in the algorithm.

Algorithm 1 Standard Multiverse Optimizer Algorithm

Require: Population size (n), Max_Iterations (It_{max})
Ensure: best solution BU

```
Initialize random solutions U_i, i = 1, 2, ..., n and iteration It ← 1
Initialize WEP_max, WEP_min, TDR and best solution BU ← NULL
Sort the solutions according to their fitness in increasing order in SU
Calculate normalized fitness NF_i for each solution U_i
while It ≤ It_max do
    Calculate Fitness for each solution U_i
    Update best solution BU
    Calculate WEP and TDR by Eq. 16 and Eq. 17.
    for each solution U_i in the population do
        Select current solution as a blackhole
        for each dimension j in the blackhole do
            if rand_1 < NF_i then
                Find other solution as whitehole using Roulette Wheel Selection form sorted universe SU
                Update blackhole(j) with SU(whitehole, j) using Eq. 14
            end if
            if rand_2 < WEP then
                if rand_3 < 0.5 then
                    blackhole(j) ← BU(j) + TDR((ub(j) − lb(j)) × rand_4 + lb(j))
                else
                    blackhole(j) ← BU(j) − TDR((ub(j) − lb(j)) × rand_4 + lb(j))
                end if
            end if
        end for
    end for
end while
return best solution BU
```

1) UNIVERSE/SOLUTION REPRESENTATION AND INITIALIZATION

The standard MVO algorithm is designed to address continuous optimization problems, whereas task scheduling falls under the category of discrete optimization problems. Thus, similar to [17], [24], we present a modified universe representation suitable for the problem of task scheduling, as shown in Fig. 3. To represent each solution or universe, we chose a one-dimensional array, with the task IDs serving as the indexes and the selected VM ID for each task as the values at each index. Further, to generate the initial set of solutions, we randomly select the VM id within the uniform distribution of [0,(m-1)], where 'm' represents the number of VMs. For better understanding and to avoid confusion, we used the words '**solution**' for each universe and '**population**' for a complete set of solutions in the rest of the paper.

2) ALGORITHM DESCRIPTION

In Algorithm 2, to start the optimization process, the proposed NIMS algorithm first accepts the population size (n), number of iterations (It_{max}), TaskList (T), and VMList (VM) as input and calculates the ETC matrix using Eqs. 1 and 2. Then, from lines 1 to 6, the algorithm performs the initial steps involved in a standard MVO optimizer. These steps include the initialization of each solution and other parameters such as WEP and TDR. Then the fitness of each solution is calculated using the fitness function described in Eq. 12, and the solution with the minimum fitness value is stored as the current best solution. After that, the solutions are sorted in ascending order of their fitness and kept in the sorted universe set SU. Further, the normalized fitness of each solution is calculated and kept in NF_i.

An additional parameter, neighborhood search probability '**NSP**' is included to support our proposed solution update

$$x_i^j = \begin{cases} \begin{cases} BU(j) + TDR((ub_j - lb_j) \times rand_4 + lb_j), & \text{if } rand_3 < 0.5 \\ BU(j) - TDR((ub_j - lb_j) \times rand_4 + lb_j), & \text{if } rand_3 \geq 0.5 \end{cases} \\ \quad \text{if } rand_2 < WEP \\ x_i^j, \quad \text{if } rand_2 \geq WEP \end{cases} \tag{15}$$

Algorithm 2 Proposed NIMS Algorithm

Require: Population size (n), Max_Iterations (It_{max}), TaskList (T), VMList (VM), ETC matrix
Ensure: best solution BU

1: Initialize random solutions U_i, $i = 1, 2, \ldots, n$ and iteration $It \leftarrow 1$
2: Initialize Empty Solution set *UpdatedUniverse* to keep updated solutions for the next iteration
3: Initialize WEP_{max}, WEP_{min}, TDR and best solution $BU \leftarrow NULL$
4: Initialize neighborhood search probability NSP_{max}, NSP_{min}
5: Sort the solutions according to their fitness in increasing order in SU
6: Calculate normalized fitness NF_i for each solution U_i
7: **while** $It \leq It_{max}$ **do**
8: Calculate Fitness for each solution U_i acc. to Eq. 12.
9: Update best solution BU
10: Calculate WEP and TDR by Eq. 16 and Eq. 17.
11: **for** each solution U_i in the population **do**
12: Perform standard MVO update on solution U_i as line 10 to 23 of Algorithm 1 and Find new solution
13: Calculate the fitness of new solution
14: **if** new solution is better than U_i **then**
15: Add new solution to *UpdatedUniverse*
16: **else**
17: Apply Algorithm 3 on the current solution U_i for fitness based neighborhood search
18: Calculate the fitness of new solution
19: **if** new solution is better than U_i **then**
20: Add new solution to *UpdatedUniverse*
21: **else**
22: Keep Original solution U_i to *UpdatedUniverse*
23: **end if**
24: **end if**
25: **end for**
26: Replace each U_i in current iteration with updated solutions in *UpdatedUniverse* for next iteration
27: **end while**
28: Return best solution BU

Algorithm 3 Neighborhood Based Updation

Require: Individual Solution (U)
Ensure: Updated solution $U^{'}$

1: Calculate neighborhood selection probability NSP using Eq. 18
2: Find Neighborhood Size NS using Eq. 19
3: **for** each dimension j in U **do**
4: **if** $rand() < NSP$ **then**
5: Find current VM id assigned for task at index j
6: Calculate the lower and upper limit of VM ids NS_l and NS_u using Eqs. 20 and 21
7: **for** each kth VM from NS_l and NS_u **do**
8: Calculate Finish Time of task j at $VM(k)$ using Eq. 22
9: Calculate the increase in Energy Consumption EC_{inc} after and before placement of task j on $VM(k)$ using Eq. 23
10: Find Fitness of $VM(k)$ using Eq. 24
11: **end for**
12: Select a Suitable VM from above using Roulette Wheel Selection for the replacement.
13: Replace current index j with newly selected VM
14: **end if**
15: **end for**
16: Return Updated Solution $U^{'}$

mechanism using the neighborhood search technique in line 4. From lines 7 to 12, the algorithm 2 repeats the optimization process similar to the steps described in algorithm 1. These steps support the standard exploration

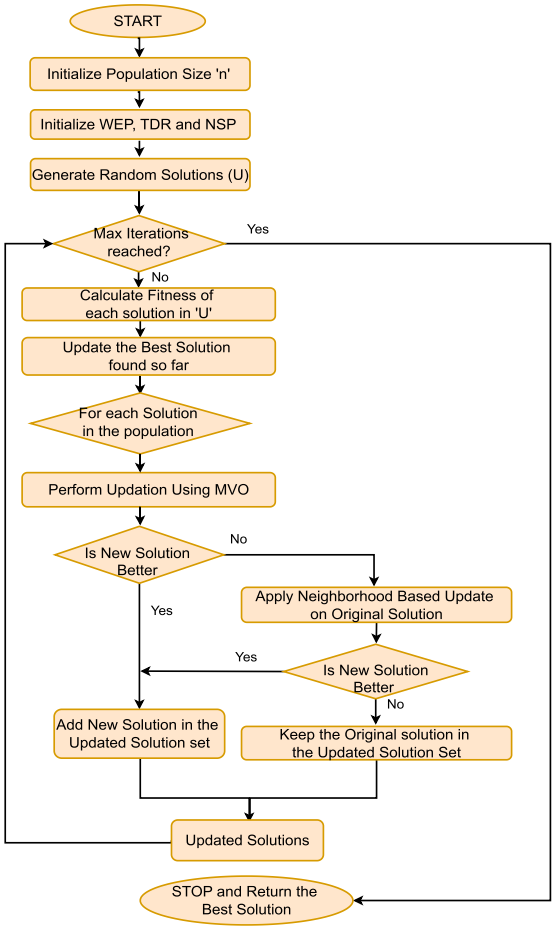

FIGURE 2. NIMS algorithm flowchart.

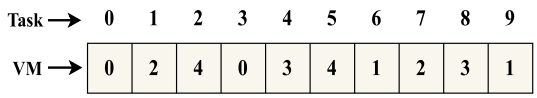

FIGURE 3. Task scheduling universe representation.

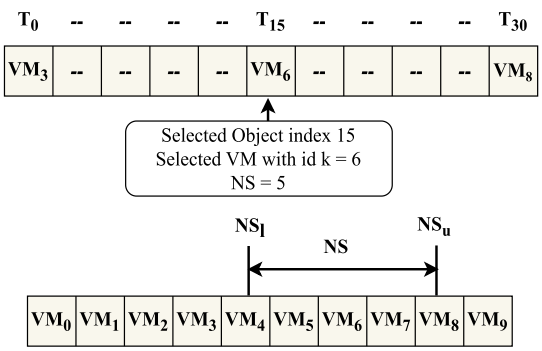

FIGURE 4. Neighborhood selection example.

and exploitation-based update of the original multiverse optimizer. However, as discussed before, the original solution update steps involved in standard MVO sometimes may not handle the combinatorial problems very effectively. Thus, a customized discrete search operator is developed in this work. Also, the proposed neighborhood-based search operator tries to select a suitable VM based on its fitness instead of random perturbation. This important update step ensures the improvement in solution quality during iterations.

In line 14, our proposed NIMS algorithm checks the fitness of the new solution found after the standard MVO update. If this solution is better than the previous solution, it is kept for the next iteration; otherwise, the proposed novel fitness-based neighborhood search is applied to the current solution. The detailed steps involved in this step are described in algorithm 3. At last, each of these steps from 11 to 26 is repeated till the maximum iteration is reached and the final best solution is returned as the result.

3) NEIGHBORHOOD BASED UPDATION

This section describes the detailed approach followed by our proposed NIMS algorithm to perform the neighborhood search-based update in the current solution, in case the standard MVO update mechanism did not perform well. The detailed steps are presented in Algorithm 3. The algorithm takes the current individual solution and returns the updated solution. In line 1, it first calculates the neighborhood search probability '*NSP*' using the following Eq. 18. The value decreases linearly to encourage exploration in the early stages and gradually shifts towards exploitation as the process progresses.

$$NSP = NSP_{max} - curr_{iter} \times \left(\frac{NSP_{max} - NSP_{min}}{max_{iter}}\right) \quad (18)$$

After that, in line 2, the neighborhood size is calculated using Eq. 19. Then, for each dimension (index) of the current solution, if the randomly generated number between [0, 1) is less than the search probability *NSP*, then the lower and upper limits of the neighborhood are calculated using Eqs. 20 and 21. This decides the range of neighbor VMs opted for further selection. Then in lines 8 and 9, the finish time of the current task and the difference in energy consumption are calculated on each of these neighbor VMs. Both of these values are combined using Eq. 24 to decide the fitness of each VM. Atlast, the roulette wheel selection is applied to select a new VM for updating the current dimension. The Roulette Wheel Selection is adopted for preserving randomness and preventing the algorithm from getting stuck in local optima. The same steps are repeated for each dimension in the current solution, and the final updated solution is returned. Further, the detailed description of each step is discussed below.

1) **Neighborhood Size Calculation:** In a similar study presented in [20], the authors compared and analyzed the effect of three different neighborhood sizes (*NS*) on the search process. They found that a very small value of *NS* could hinder the algorithm's efficient neighborhood exploration, while a high value could trap the algorithm in the local optima and possibly increase the algorithm's computational complexity. Following this, in our work, we selected an adaptive value of *NS* that changes with the number of iterations and VMs to control the search process. According to

Eq. 19, the calculation of NS is done.

$$NS = round(2(c_1 e^{\frac{-c_2}{m}})(1 - e^{\frac{-c_3}{It_{max}}}) + 1) \quad (19)$$

where m denotes the number of VMs, and It_{max} denotes the maximum number of iterations. c_1, c_2, and c_3 are the constants to control the size of the neighborhood, and their values are selected as $c_1 = 8$, $c_2 = 15$, and $c_3 = 10$, according to [20].

2) **Neighborhood Boundaries Calculation:** The neighborhood VMs for the selected dimension are determined within the lower and upper limits of the VM IDs, denoted as NS_l and NS_u. These limits are calculated using Eqs. 20 and 21.

$$NS_l = ceil[VM(k) - \frac{NS}{2}] \quad (20)$$

$$NS_u = floor[VM(k) + \frac{NS}{2}] \quad (21)$$

where $VM(k)$ represents the VM with id k selected for update. The values of NS_l and NS_u must be within the range of the number of VMs such that $0 \leq (NS_l, NS_u) \leq (m - 1)$. Fig. 4 displays an illustrative example for the boundary calculation for a universe where the selected object is at index 15, VM id $k = 6$, and neighborhood size $NS = 5$.

3) **Fitness Calculation for Each VM in the Neighborhood:** After selecting the neighbor VMs for dimension 'j' Algorithm 3 calculates the fitness for each $VM(k)$ present in the range from NS_l to NS_u. The following two factors determine the fitness of a virtual machine (VM):

- The **first factor** is the expected finish time of the task $T(j)$ on $VM(k)$. To calculate this, we use Eq. 22.

$$EFT(VM(k)) = VM_{finish}(k) + ETC_{j,k} \quad (22)$$

where $VM_{finish}(k)$ is the current finish time of $VM(k)$ with already assigned tasks. This value is calculated using Eq. 3, and $ETC_{j,k}$ is the expected time to compute task $T(j)$ on $VM(k)$, which is calculated using Eq. 1.

- And the **second factor** is an increase in the overall energy consumption if the task $T(j)$ is placed on $VM(k)$. We calculate this value by measuring the difference between the energy consumption of all VMs after mapping $T(j)$ on $VM(k)$ and the energy consumption prior to it. The mathematical formula for the same is given in Eq. 23.

$$EC_{inc} = EC_{afterplacement} - EC_{beforeplacement} \quad (23)$$

where EC_{inc} is the increased energy consumption. The values for $EC_{afterplacement}$ and $EC_{beforeplacement}$ are calculated using Eq. 5.

Further, the final fitness of each $VM(k)$ is calculated using Eq. 24. The reversed value of each factor is taken to assign a higher fitness value to the VM, which can finish the task in less time and cause a lesser increase in energy consumption. Also, the normalized values are used to bring them to the same scale. We selected equal 0.5 weights to give equal priority to each factor.

$$VM_{fit}(k) = 0.5 \times Rev(EFT(VM_k)) + 0.5 \times Rev(EC_{inc}) \quad (24)$$

a: TIME COMPLEXITY

The computational complexity of our proposed NIMS algorithm is influenced by the complexity of the underlying multiverse optimizer and the additional time taken by the neighborhood search process. In this section, we present the worst-case time complexity of our proposed algorithm.

The total time complexity of the original MVO algorithm is described in paper [41] as follows:

$$O(MVO) = O(I(n^2 + n \times d \times logn))$$

where, I, is the number of iterations, n^2 is the worst-case time complexity of the Quick-Sort algorithm applied for sorting the universes, and $(n \times d \times logn)$ is the time complexity of the roulette wheel used in every iteration for each solution and each dimension.

Further, the proposed neighborhood search-based update is applied to only selected solutions. But we consider the worst-case time complexity here.

1) The neighborhood Size Calculation is a unit-time operation within each iteration. Thus, it takes $O(I)$.
2) The neighborhood boundary is a unit-time operation calculated for each dimension within a solution. So, the worst-case time for this operation is $O(I \times n \times d)$.
3) The fitness calculation for each VM within the boundary takes $O(I \times n \times d \times NS \times m)$, where NS is the neighborhood size and m is the VM count to calculate the energy consumption.

Thus, the total time complexity of the complete NIMS algorithm is:

$$O(NIMS) = O(I(n^2 + n \times d \times logn + n \times d \times NS \times m))$$

For simplicity it can also be rewritten as:

$$O(NIMS) = O(I \times n(n + d(logn + NS \times m)))$$

TABLE 4. Experimental environment.

Cloud Entity	Parameter	Value
Data center	Number of data centers	1
Physical Machine (Host)	Number of hosts	15
	Storage Capacity	1 TB
	Bandwidth	10 GB/s
	VM allocation and scheduling policy	Best-Fit Decreasing (Time-Shared)
Virtual Machine	Number of VMs	20-100
	Storage Capacity	20 GB
	Bandwidth	1 GB/s
	VMM	Xen
	Task scheduling Policy	Space-Shared

TABLE 5. Hosts' configuration.

Host Types	Parameter	Value
HP ProLiant, DL585 G5 AMD Opteron 8384	CPU Capacity	2700 MIPS
	RAM	16 GB
	Cores	16
HP ProLiant, DL785 G5 AMD Opteron 8381 HE	CPU Capacity	2500 MIPS
	RAM	32 GB
	Cores	32

TABLE 6. VMs' configurations.

VM Types	Parameter	Value
Type 1	CPU Capacity	2600 MIPS
	RAM	0.85 GB
	Cores	1
	Max Power Consumption	375 watts
	Idle Power Consumption	150 watts
Type 2	CPU Capacity	2000 MIPS
	RAM	3.75 GB
	Cores	1
	Max Power Consumption	300 watts
	Idle Power Consumption	120 watts
Type 3	CPU Capacity	1000 MIPS
	RAM	1.7 GB
	Cores	1
	Max Power Consumption	165 watts
	Idle Power Consumption	66 watts
Type 4	CPU Capacity	500 MIPS
	RAM	613 MB
	Cores	1
	Max Power Consumption	80 watts
	Idle Power Consumption	32 watts

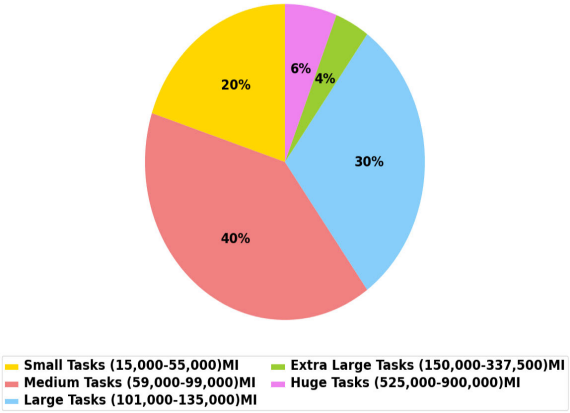

FIGURE 5. GoCJ task distribution.

V. EXPERIMENTAL SETUP AND WORKLOAD DESCRIPTION

This section provides a detailed description of the experimental setup and workload types considered for evaluation. The experimental setup comprises information about the underlying cloud entities and the resource characteristics.

A. EXPERIMENTAL SETUP

To conduct the experiments, we have used the CloudSim toolbox [43], which is a globally acknowledged and extensively employed cloud simulator. CloudSim is an open-source framework written in Java, created for the purpose of developing and simulating different scheduling and allocation policies. The framework facilitates the creation of various cloud entities, including data centers, hosts, virtual machines, and cloudlets (tasks). Additionally, we have the capability to incorporate a range of scheduling algorithms and evaluate their performance using CloudSim. The simulation is done on a Windows 10 (64-bit) machine, equipped with an Intel Core i5 processor (1.70 GHz), 8 GB of RAM, a 1 TB hard disk, and the Eclipse Java Development Environment. Our experimental setup assumes a single data center with 15 heterogeneous physical servers (hosts). The number of virtual machines varies according to different experimental scenarios. Table 4 presents the experimental environment. Tables 5 and 6 display the configuration of different types of hosts' [49] and VMs' [44] considered for evaluation. We have taken these configurations form SPEC benchmark dataset.

B. WORKLOAD DESCRIPTION

To test the algorithms on realistic workload traces, we consider three widely adopted benchmark datasets to generate cloud tasks. These datasets are the workload trace logs from Google, HPC2N, and NASA-Ames-iPSC/860 clusters. The primary reason for choosing these datasets over synthetic tasks is that they reflect the realistic cloud workload in terms of size and characteristics. A detailed description of these datasets is provided below.

1) GOOGLE CLOUD JOBS (GOCJ) DATASET

Hussain and Aleem [45] have developed the GoCJ dataset that serves as a standardized workload for assessing different scheduling and resource provisioning policies in the cloud infrastructure. The dataset captures the true workload behavior patterns observed in Google cluster traces and logs from the M45 supercomputing system. For evaluation purposes, the authors employed the Monte Carlo simulation method to produce the dataset, which is publicly available in the Mendeley Data Repository [46]. The dataset is a collection of text files representing workloads of different sizes. Each file contains numeric values representing the size of cloud tasks in million instructions (MI). Fig. 5 shows the statistical distribution of these tasks.

2) HPC2N AND NASA AMES IPSC/860 PARALLEL WORKLOAD LOGS

The HPC2N (High-Performance Computing Centre North) parallel workload logs are provided by Ake Sandgren and Michael Jack, while the logs from NASA Ames iPSC/860 are presented by Bill Nitzberg. Both the logs are made publicly available in the Parallel Workload Archive [47]. The HPC2N log is collected from a 120-node Linux cluster located in Sweden, whereas the NASA iPSC/860 log is taken from a 128-node iPSC/860 hypercube cluster in the NASA Ames research center. The specification of both workloads is presented in Table 7. At last, a comparative analysis of the proposed algorithm with other state-of-the-art approaches is also done under several experimental scenarios.

TABLE 7. Specification for parallel workload traces.

Workload Name	Log Duration	Cluster Size	Task Count	User Count	Files
NASA iPSC	Oct 93 - Dec 93	128 CPU	18,239	69	NASA-iPSC-1993-3.1-cln.swf
HPC2N Seth log	Jul 2002- Jan 2006	240 CPU	202,871	257	HPC2N-2002-2.2-cln.swf

TABLE 8. Optimization parameters.

Algorithm	Parameter	Value
PSO Based Algorithms	Adaptive Inertia Weight (w)	[0.9 to 0.4]
	Acceleration Coefficients (c1,c2)	[1.49,1.49]
	Opposition Probability for OPSO	0.3
MVO Based Algorithms	Wormhole Existence Probability (WEP)	[1.0 to 0.2]
	Neighborhood Search Probability (NSP)	[0.7 to 0.1]
GWO Based Algorithms	GWO coefficient vector a	[2.0 to 0]
GA Based Algorithms	Crossover and Mutation Probability	[0.7, 0.8]
	Mutation Probability	[0.05, 0.1]
FP Based Algorithms	Switch Probability	0.8
	lambda	1.5

VI. EXPERIMENTS AND RESULTS

This section presents the experimental implementation and performance evaluation of the proposed NIMS algorithm with seven state-of-the-art metaheuristic-based scheduling algorithms, including EMVO [13], IMOMVO [14], OPSO [16], LJFPPSO [17], TSIGA [35], FPGWO [36], and MVO [41]. These algorithms are selected based on their proven effectiveness towards solving task scheduling optimization problems. Another reason to select these algorithms is that they are recently developed and utilize various other improvisation techniques to improve the performance of their algorithms. Our comparison majorly focuses on evaluation of five performance metrics: makespan, energy consumption, throughput, degree of imbalance, and average resource utilization. The detailed description of these metrics is made available in Section III. Further, all the algorithms are re-implemented and tested under the same physical infrastructure and resource configurations for valid and fair comparison. To evaluate the algorithms' performance on varying workload sizes and resource capabilities, we have designed the following test cases.

1) To evaluate the performance of each algorithm on increasing workload size and fixed number of resources, we have conducted experiments taking task count ranging from 100-600 from the GoCJ dataset and 500-2000 from HPC2N and NASA workload logs. The number of VMs is kept fixed to 80.
2) To evaluate the performance on fixed workload size while changing the number of VMs, we have taken 500 tasks from the GoCJ dataset and 1000 tasks from the HPC2N and NASA workload logs, whereas the VM count varies from 20 to 100.

Each experiment is conducted 10 times for each algorithm to ensure statistical significance of the attained results. Table 8 displays the selected optimization parameters for each of the considered algorithms aligned with their respective papers. However, the population size and maximum number of iterations were fixed to 50 and 100, respectively. The following subsection discusses the performance analysis for each metric in detail.

A. PERFORMANCE ANALYSIS USING MAKESPAN

Figs. 6 to 8 present the achieved makespan values by the proposed NIMS and other comparative algorithms under various experimental conditions described in the previous section. A lower makespan indicates superior performance, while a higher makespan denotes inferior performance. The algorithms show an increasing trend in the makespan values with increased task count over a fixed number of VMs. This increment is obvious due to the increased workload on VMs. However, on the other side, there is a decreasing trend in the makespan values while assigning a fixed number of tasks over a variable VM count. This decrease is achieved due to the reduction of VM load with the increased VM count. Further, the results also demonstrate that the proposed NIMS algorithm achieves a superior makespan value over other comparative algorithms across almost all test conditions and datasets. The corresponding numerical values are displayed in Tables 9 and 10. Finally, the performance improvement of the proposed NIMS algorithm ranges from 9-49 % over EMVO, 23-56 % over IMOMVO, 29-51 % over OPSO, 4-45 % over LJFPPSO, 21-52 % over TSIGA, 1-16 % over FPGWO, and 26-49 % over MVO across all datasets. This improvement is because of the improvement adopted in the solution update mechanism of MVO using fitness-based neighborhood search. This search allows the selection of a fitter possible VM with a smaller task's finish time and the least increase in energy consumption for tasks while updating individual solutions in the process of optimization.

B. PERFORMANCE ANALYSIS USING TOTAL ENERGY CONSUMPTION

To enhance the precision of observed results, the energy consumption attained by all the algorithms is recorded in kilowatt-hour (kWh) units and displayed in Figs. 9 to 11. The plots reveal the efficacy of the proposed NIMS algorithm over other comparative approaches, with significant reductions in total energy consumption for the generated schedule across all the datasets for all the experiments. The associated numerical values presented in Tables 11 and 12 further demonstrate that the NIMS algorithm achieved a substantial reduction in energy consumption over EMVO, IMOMVO, OPSO, LJFPPSO, TSIGA, FPGWO, and MVO by approximately 7-34 %, 17-42 %, 18-37 %, 2-29 %, 15-40 %, 1-9 %, and 9-36 % respectively across all the workloads. This reduction is possibly achieved because the increase in the energy consumption is considered as an important factor while choosing a new VM for the given

TABLE 9. Makespan (minutes) for variable number of tasks and fixed VM count.

Dataset	Task Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	100	6.37	7.56	8.14	6.05	7.33	5.83	6.59	5.77
	200	12.71	15.32	14.26	6.77	13.56	8.48	12.79	7.75
	300	17.65	23.16	21.01	9.32	19.98	12.39	17.57	10.41
	400	22.32	27.75	23.25	14.36	23.82	14.31	21.34	12.61
	500	28.72	35.61	30.43	17.96	31.59	17.15	27.73	15.7
	600	35.7	43.3	35.82	21.91	38.52	21.39	35.54	19.14
HPC2N	500	19.55	24.16	19.34	16.35	22.66	12.02	20.51	11.14
	1000	39.21	45.92	37.10	33.49	43.94	23.72	40.02	21.85
	1500	58.51	67.96	55.77	52.30	63.33	37.15	60.89	33.91
	2000	78.45	91.75	76.81	72.12	82.64	52.66	77.05	45.18
NASA	500	1.03	1.21	0.97	0.81	1.14	0.62	1.02	0.56
	1000	2.36	2.71	2.18	1.89	2.55	1.39	2.37	1.26
	1500	3.64	4.28	3.36	3.05	3.96	2.31	3.57	2.02
	2000	4.62	5.44	4.39	4.17	5.12	3.13	4.72	2.65

TABLE 10. Makespan (minutes) for variable number of VMs and fixed task count.

Dataset	VM Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	20	58.93	85.63	60.73	63.6	66.3	41.58	55.57	40.81
	40	39.8	49.59	40.25	40.37	42.85	26.31	39.87	25.02
	60	32.96	39.99	33.67	24.49	36.5	20.85	33.78	18.81
	80	28.72	35.61	30.43	17.96	31.59	17.15	27.73	15.7
	100	25.88	30.04	27.83	14.2	28.9	16.17	26.85	13.6
HPC2N	20	98.96	143.01	101.76	107.20	116.61	68.22	99.65	62.5
	40	62.22	77.50	58.21	65.84	70.44	39.70	62.18	36.24
	60	46.16	56.14	45.37	46.01	53.76	30.15	46.75	26.97
	80	39.21	45.92	37.10	33.49	43.94	23.72	40.02	21.85
	100	34.62	39.21	32.53	26.61	37.71	20.44	34.32	18.97
NASA	20	5.43	7.91	5.84	6.29	6.26	3.89	5.60	3.50
	40	3.65	4.48	3.43	3.65	4.02	2.23	3.74	2.03
	60	2.77	3.43	2.55	2.55	2.92	1.74	2.92	1.51
	80	2.36	2.71	2.18	1.89	2.55	1.39	2.37	1.26
	100	2.13	2.45	1.93	1.50	2.19	1.24	2.04	1.08

TABLE 11. Energy consumption (kWh) for variable number of tasks and fixed VM count.

Dataset	Task Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	100	1.12	1.26	1.34	1.07	1.23	1.04	1.15	1.04
	200	2.24	2.57	2.44	1.51	2.35	1.71	2.26	1.63
	300	3.22	3.90	3.63	2.18	3.51	2.55	3.21	2.31
	400	4.06	4.73	4.17	3.07	4.25	3.05	3.95	2.85
	500	5.18	6.03	5.40	3.85	5.53	3.74	5.07	3.56
	600	6.49	7.42	6.50	4.78	6.83	4.70	6.47	4.42
HPC2N	500	3.58	4.15	3.56	3.19	3.96	2.64	3.71	2.53
	1000	7.16	7.99	6.90	6.45	7.74	5.23	7.26	4.99
	1500	10.72	11.89	10.39	9.96	11.32	8.08	11.02	7.66
	2000	14.36	16.01	14.17	13.59	14.89	11.18	14.19	10.22
NASA	500	0.18	0.21	0.18	0.16	0.20	0.13	0.19	0.12
	1000	0.43	0.47	0.41	0.37	0.45	0.31	0.43	0.29
	1500	0.66	0.74	0.63	0.59	0.70	0.50	0.66	0.46
	2000	0.85	0.95	0.82	0.79	0.91	0.66	0.86	0.61

task while updating a solution using the novel fitness based neighborhood search mechanism.

C. PERFORMANCE ANALYSIS USING THROUGHPUT

System throughput represents the capacity of executing the maximum possible number of tasks in a unit time. An efficient scheduling algorithm is desired to have high throughput value. Thus, one of the crucial objectives of our work is to compare the performance of all the algorithms in terms of the throughput achieved. Fig. 12 to 14 displays the graphical representation of throughput values attained by all the algorithms. Each figure represents the result for a different experimental setup and dataset. The efficacy of the proposed NIMS algorithm is clearly observed from the figures over other comparative algorithms. Although, in the case of variable tasks from the GoCJ dataset the The LJFPPSO algorithm initially demonstrates better throughput for a smaller number of tasks, but its performance degrades as the problem size increases. This decline is likely due to the largest-job-to-fastest-processor heuristic, which depends on the ratio of large and small tasks in the dataset. Tables 13 and 14 present the corresponding numerical values of

TABLE 12. Energy consumption (kWh) for variable number of VMs and fixed task count.

Dataset	VM Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	20	3.46	4.29	3.52	3.60	3.68	2.91	3.35	2.88
	40	4.1	4.71	4.13	4.13	4.28	3.24	4.11	3.17
	60	4.69	5.34	4.76	3.91	5.03	3.55	4.77	3.36
	80	5.18	6.03	5.40	3.85	5.53	3.74	5.07	3.56
	100	5.63	6.27	5.93	3.82	6.10	4.11	5.78	3.72
HPC2N	20	5.38	6.73	5.51	5.63	5.91	4.41	5.40	4.22
	40	6.16	7.10	5.91	6.38	6.67	4.75	6.16	4.53
	60	6.59	7.51	6.52	6.58	7.29	5.10	6.65	4.79
	80	7.16	7.99	6.90	6.45	7.74	5.23	7.29	4.99
	100	7.65	8.36	7.33	6.42	8.13	5.45	7.61	5.22
NASA	20	0.30	0.38	0.31	0.33	0.33	0.25	0.31	0.24
	40	0.36	0.41	0.35	0.36	0.38	0.27	0.37	0.26
	60	0.39	0.45	0.37	0.37	0.41	0.29	0.41	0.27
	80	0.43	0.47	0.41	0.37	0.45	0.31	0.43	0.29
	100	0.46	0.51	0.43	0.37	0.47	0.33	0.45	0.3

TABLE 13. Throughput for variable number of tasks and fixed VM count.

Dataset	Task Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	100	0.26	0.22	0.20	0.27	0.23	0.28	0.25	0.29
	200	0.26	0.21	0.23	0.49	0.24	0.39	0.26	0.43
	300	0.28	0.21	0.23	0.53	0.25	0.40	0.28	0.48
	400	0.30	0.24	0.28	0.46	0.27	0.46	0.31	0.52
	500	0.29	0.23	0.27	0.46	0.26	0.48	0.30	0.53
	600	0.28	0.23	0.28	0.46	0.26	0.46	0.28	0.52
HPC2N	500	0.43	0.34	0.43	0.51	0.36	0.69	0.4	0.74
	1000	0.42	0.36	0.45	0.5	0.37	0.7	0.42	0.76
	1500	0.42	0.37	0.44	0.48	0.39	0.67	0.41	0.73
	2000	0.42	0.36	0.43	0.46	0.4	0.63	0.43	0.73
NASA	500	8.09	6.87	8.54	10.26	7.33	13.4	8.23	14.86
	1000	7.06	6.17	7.63	8.78	6.55	12.02	7.08	13.16
	1500	6.88	5.85	7.44	8.19	6.3	10.82	7.00	12.36
	2000	7.22	6.13	7.6	7.97	6.5	10.66	7.07	12.57

TABLE 14. Throughput for variable number of VMs and fixed task count.

Dataset	VM Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	20	0.14	0.09	0.13	0.13	0.12	0.2	0.15	0.2
	40	0.21	0.17	0.21	0.2	0.19	0.31	0.21	0.33
	60	0.25	0.21	0.25	0.34	0.23	0.4	0.25	0.44
	80	0.29	0.23	0.27	0.46	0.26	0.48	0.3	0.53
	100	0.32	0.27	0.3	0.59	0.29	0.51	0.31	0.61
HPC2N	20	0.16	0.11	0.16	0.15	0.14	0.24	0.16	0.26
	40	0.26	0.21	0.28	0.25	0.23	0.42	0.26	0.46
	60	0.36	0.29	0.36	0.36	0.31	0.55	0.36	0.61
	80	0.42	0.36	0.45	0.5	0.37	0.7	0.42	0.76
	100	0.48	0.42	0.51	0.63	0.44	0.81	0.49	0.88
NASA	20	3.06	2.11	2.86	2.65	2.67	4.3	2.98	4.76
	40	4.56	3.73	4.86	4.57	4.16	7.46	4.46	8.19
	60	6.01	4.85	6.57	6.52	5.72	9.59	5.71	10.99
	80	7.06	6.17	7.63	8.78	6.55	12.02	7.08	13.16
	100	7.84	6.8	8.71	11.1	7.63	13.4	8.22	15.35

the throughput for each algorithm. Our proposed NIMS algorithm is able to attain up to 90%, 83%, and 95% increase in throughput over EMVO for GoCJ, HPC2N, and NASA datasets, respectively. The increases in throughput over IMOMVO are up to 130%, 136%, and 126%. Further, over OPSO, the improvement is noticed as 108%, 72%, and 76% for all three datasets. In the case of LJFPPSO, our algorithm could achieve a maximum improvement of 84% among all the datasets. Similarly, maximum 110% improvement over TSIGA, 20% over FPGWO and 96% over MVO is noticed in throughput value achieved by our proposed algorithm. One of the possible reasons behind this improvement is the reduced value of makespan attained by the proposed NIMS algorithm.

D. PERFORMANCE ANALYSIS USING DEGREE OF IMBALANCE

Another important factor considered in the efficacy of a scheduling algorithm is its capability of uniform workload distribution amongst available resources. A uniform load distribution further maximizes the overall resource utilization and also helps in reduction of the static (idle time) energy

TABLE 15. Degree of imbalance for variable number of tasks and fixed VM count.

Dataset	Task Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	100	3.41	3.85	3.90	3.32	3.84	4.27	3.57	3.78
	200	3.00	3.62	3.35	1.37	3.15	2.62	3.06	2.17
	300	2.61	3.32	3.07	1.11	2.91	2.37	2.64	1.86
	400	2.47	3.05	2.58	1.29	2.57	1.95	2.35	1.65
	500	2.46	3.02	2.63	1.21	2.69	1.72	2.38	1.46
	600	2.36	2.93	2.39	1.10	2.49	1.65	2.38	1.40
HPC2N	500	2.27	2.8	2.24	1.54	2.49	1.63	2.36	1.43
	1000	2.08	2.41	2.03	1.47	2.22	1.4	2.13	1.2
	1500	1.99	2.33	1.9	1.52	2	1.26	2.06	1.14
	2000	1.97	2.33	1.92	1.57	1.92	1.31	1.92	1.1
NASA	500	2.37	2.78	2.3	1.56	2.58	1.75	2.31	1.45
	1000	2.16	2.56	2.09	1.45	2.32	1.37	2.15	1.2
	1500	2.06	2.49	1.9	1.48	2.17	1.35	2.04	1.14
	2000	1.97	2.36	1.91	1.54	2.07	1.35	2.02	1.11

TABLE 16. Degree of imbalance for variable number of VMs and fixed task count.

Dataset	VM Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	20	1.02	1.67	1.11	1.15	1.14	0.49	0.94	0.5
	40	1.55	2.07	1.69	1.6	1.73	1.07	1.61	0.9
	60	2.08	2.5	2.11	1.25	2.2	1.47	2.14	1.26
	80	2.46	3.02	2.63	1.21	2.69	1.72	2.38	1.46
	100	2.81	3.25	2.95	1.28	3.05	2.17	2.9	1.79
HPC2N	20	1.11	1.72	1.18	1.25	1.38	0.62	1.16	0.52
	40	1.58	2.02	1.45	1.55	1.71	0.91	1.56	0.81
	60	1.82	2.2	1.83	1.57	2.03	1.17	1.81	1
	80	2.08	2.41	2.03	1.47	2.22	1.4	2.13	1.2
	100	2.37	2.72	2.28	1.47	2.4	1.51	2.34	1.39
NASA	20	1.06	1.65	1.15	1.28	1.26	0.59	1.08	0.49
	40	1.55	1.99	1.5	1.52	1.73	0.9	1.61	0.75
	60	1.87	2.35	1.76	1.51	1.95	1.19	1.95	0.97
	80	2.16	2.56	2.09	1.45	2.32	1.37	2.15	1.21
	100	2.47	2.92	2.36	1.43	2.54	1.63	2.41	1.39

TABLE 17. Average resource utilization rate for variable number of tasks and fixed VM count.

Dataset	Task Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	100	0.29	0.26	0.25	0.30	0.26	0.23	0.28	0.26
	200	0.33	0.27	0.30	0.57	0.31	0.38	0.33	0.46
	300	0.38	0.30	0.32	0.65	0.34	0.42	0.38	0.53
	400	0.39	0.32	0.38	0.58	0.37	0.50	0.42	0.56
	500	0.39	0.32	0.37	0.59	0.36	0.53	0.41	0.58
	600	0.40	0.33	0.40	0.61	0.38	0.55	0.41	0.60
HPC2N	500	0.42	0.35	0.43	0.51	0.38	0.56	0.41	0.6
	1000	0.43	0.39	0.455	0.51	0.4	0.6	0.43	0.63
	1500	0.44	0.39	0.47	0.5	0.43	0.61	0.44	0.63
	2000	0.45	0.39	0.46	0.5	0.44	0.6	0.46	0.64
NASA	500	0.41	0.35	0.42	0.52	0.37	0.54	0.41	0.6
	1000	0.42	0.37	0.45	0.52	0.4	0.61	0.43	0.63
	1500	0.44	0.38	0.47	0.52	0.41	0.6	0.45	0.64
	2000	0.45	0.39	0.47	0.5	0.42	0.59	0.45	0.64

consumption of resources. In this view, the degree of imbalance is an important metric that represents the degree of variation in the load distribution amongst all VMs. The lower value of DoI indicates a better distribution of workload. Figs. 15, 16, and 17 display the results attained by our proposed NIMS algorithm and other comparative approaches in terms of degree of imbalance. It is inferred that the proposed algorithm attains superior performance over other algorithms for HPC2N and NASA workloads. However, in the case of GoCJ, the LJFPPSO delivers better load distribution for some experiments. One possible reason behind this is that the LJFPPSO algorithm is influenced by another heuristic algorithm named LJFP (Longest Job to Fastest Processor), which first allocates the longest jobs on the highest capacity VM. Also, from Fig. 5 we can see that in the GoCJ dataset the very small portion of the dataset is occupied by heavy jobs. Thus, the algorithm could achieve a better degree of imbalance for the GoCJ dataset. Further, the quantitative values associated with the same are presented in Tables 15 and 16. From these tables, we can clearly observe that in the case of the GoCJ dataset, the suggested NIMS algorithm is able to reduce the degree of imbalance up to

TABLE 18. Average resource utilization rate for variable number of VMs and fixed task count.

Dataset	VM Count	EMVO	IMOMVO	OPSO	LJFPPSO	TSIGA	FPGWO	MVO	NIMS
GoCJ	20	0.71	0.53	0.69	0.66	0.62	0.87	0.73	0.88
	40	0.55	0.46	0.55	0.54	0.51	0.68	0.54	0.73
	60	0.45	0.38	0.44	0.59	0.43	0.59	0.44	0.65
	80	0.39	0.32	0.37	0.59	0.36	0.53	0.41	0.58
	100	0.35	0.3	0.34	0.58	0.32	0.45	0.34	0.53
HPC2N	20	0.64	0.49	0.62	0.60	0.55	0.81	0.64	0.85
	40	0.53	0.44	0.57	0.51	0.49	0.72	0.54	0.75
	60	0.48	0.41	0.49	0.5	0.43	0.64	0.48	0.68
	80	0.43	0.39	0.45	0.51	0.41	0.61	0.43	0.63
	100	0.39	0.35	0.42	0.51	0.38	0.57	0.4	0.59
NASA	20	0.67	0.51	0.65	0.60	0.60	0.83	0.66	0.88
	40	0.53	0.45	0.56	0.53	0.5	0.73	0.52	0.78
	60	0.48	0.4	0.51	0.52	0.46	0.65	0.46	0.7
	80	0.42	0.37	0.45	0.52	0.4	0.61	0.43	0.63
	100	0.38	0.33	0.4	0.52	0.37	0.54	0.39	0.59

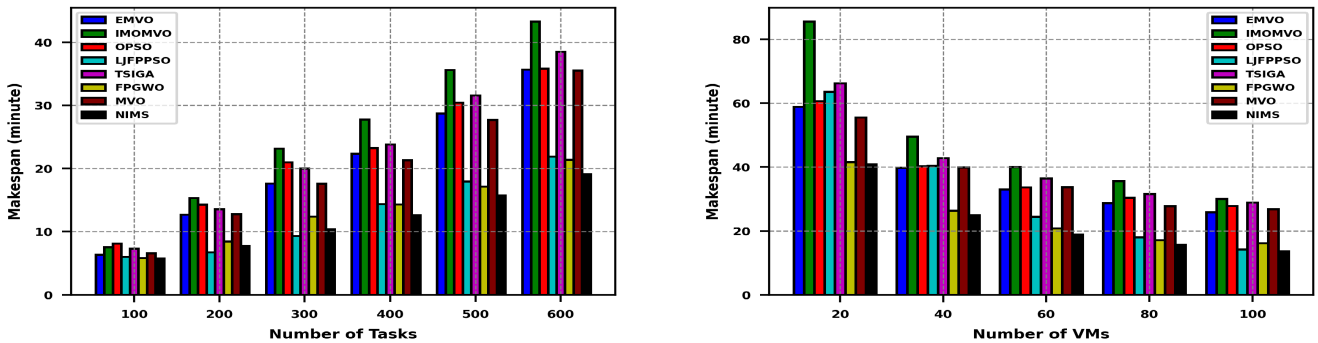

FIGURE 6. Makespan vs. Number of tasks and number of VMs using GoCJ dataset.

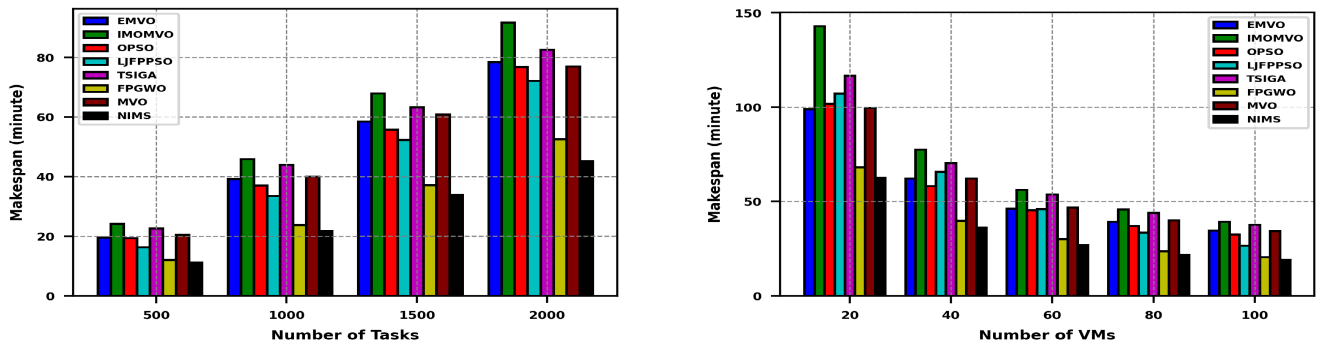

FIGURE 7. Makespan vs. Number of tasks and number of VMs using HPC2N dataset.

50% over EMVO, 70% over IMOMVO, 54% over OPSO, 56% over TSIGA, 21% over FPGWO, and 46% over MVO algorithms. However, there is an increase of up to 67% over LJFPPSO. In the case of the HPC2N dataset, the NIMS algorithm is able to achieve up to 53%, 69%, 56%, 58%, 62%, 16%, and 55% over EMVO, IMOMVO, OPSO, LJFPPSO, TSIGA, FPGWO, and MVO, respectively. It can also be observed that, for the NASA dataset, our NIMS algorithm outperforms EMVO, IMOMVO, OPSO, LJFPPSO, TSIGA, FPGWO, and MVO by 53%, 70%, 57%, 61%, 61%, 18%, and 54%, respectively.

E. PERFORMANCE ANALYSIS USING AVERAGE RESOURCE UTILIZATION RATE

Finally, we compared the introduced NIMS algorithm against all the other comparative algorithms to measure the effect on average resource utilization percentage. From Fig. 18 to 20, the attained values of average resource utilization rates (ARUR) are plotted for each algorithm per dataset. The numerical value supporting the figures is also mentioned in Tables 17 and 18. As the higher value of resource utilization is desired by a good scheduling algorithm, we can see the superiority of our suggested approach over other algorithms

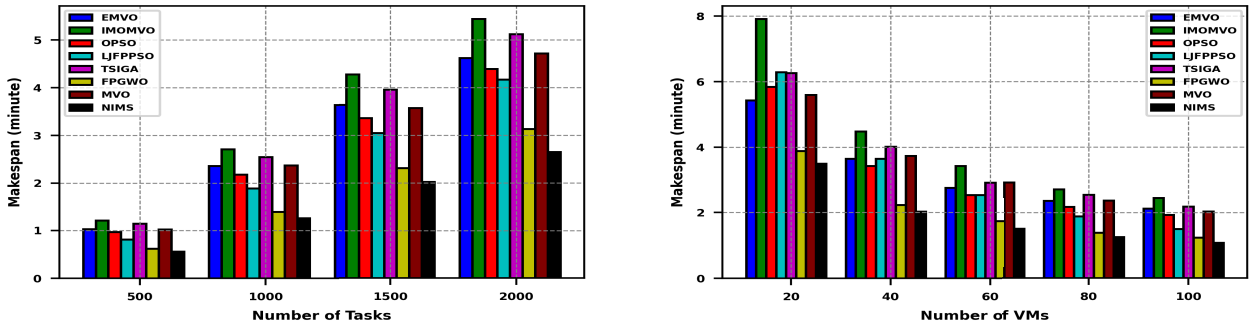

FIGURE 8. Makespan vs. Number of tasks and number of VMs using NASA-iPSC dataset.

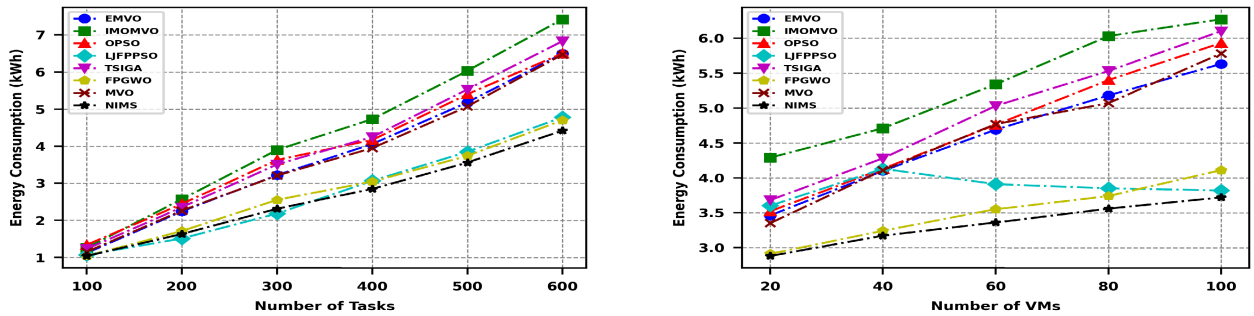

FIGURE 9. Energy consumption vs. Number of tasks and number of VMs using GoCJ dataset.

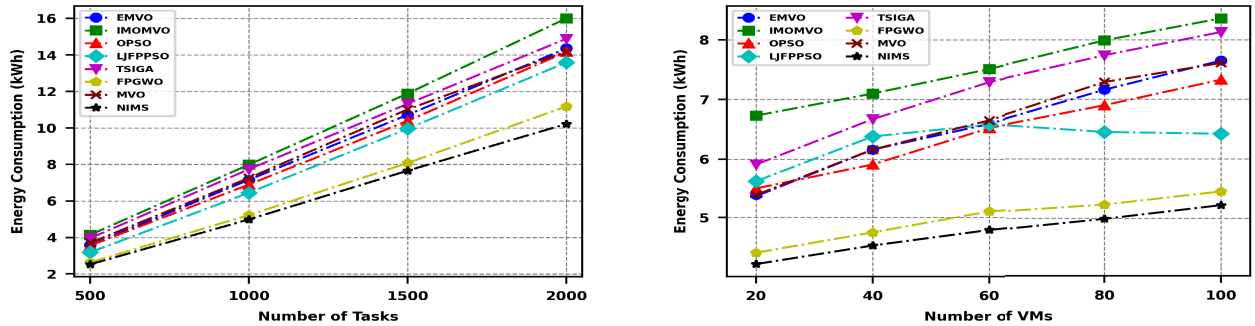

FIGURE 10. Energy consumption vs. Number of tasks and number of VMs using HPC2N dataset.

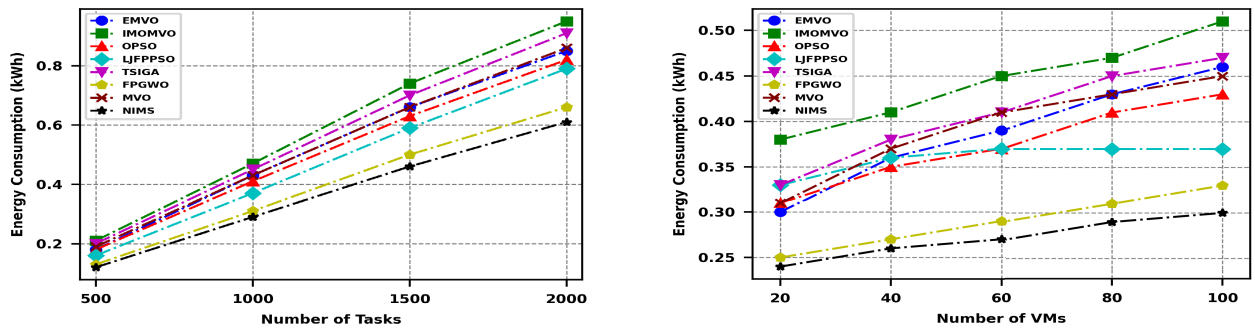

FIGURE 11. Energy consumption vs. Number of tasks and number of VMs using NASA-iPSC dataset.

for each of the three datasets. However, in case of variable number of tasks from the GoCJ dataset, the LJFPPSO algorithm performs little better than the proposed algorithm, and the possible reason behind it is the better load distribution discussed in the previous section. The recorded percentage improvement of the proposed NIMS algorithm in terms of ARUR is up to 51% over EMVO, 81% over IMOMVO, 65% over OPSO, 35% over LJFPPSO (over variable VM

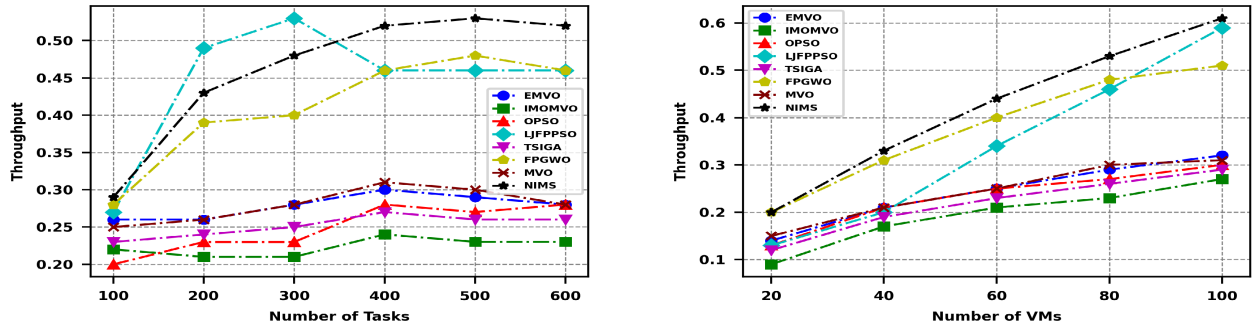

FIGURE 12. Throughput vs. Number of tasks and number of VMs using GoCJ dataset.

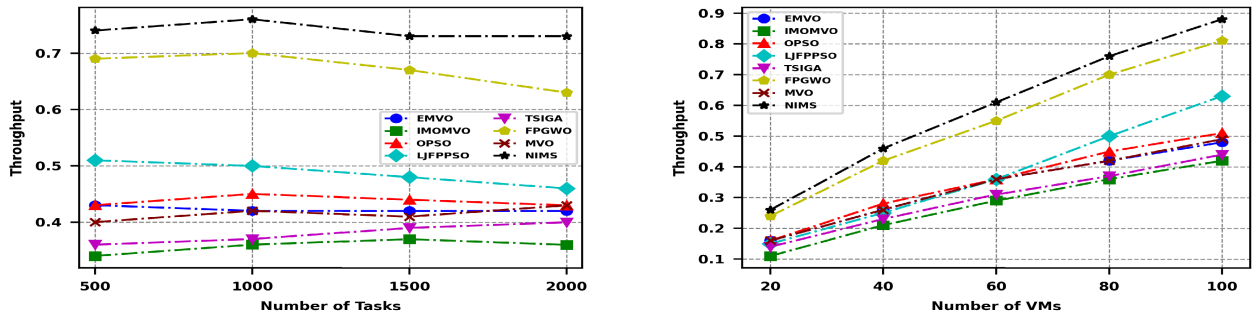

FIGURE 13. Throughput vs. Number of tasks and number of VMs using HPC2N dataset.

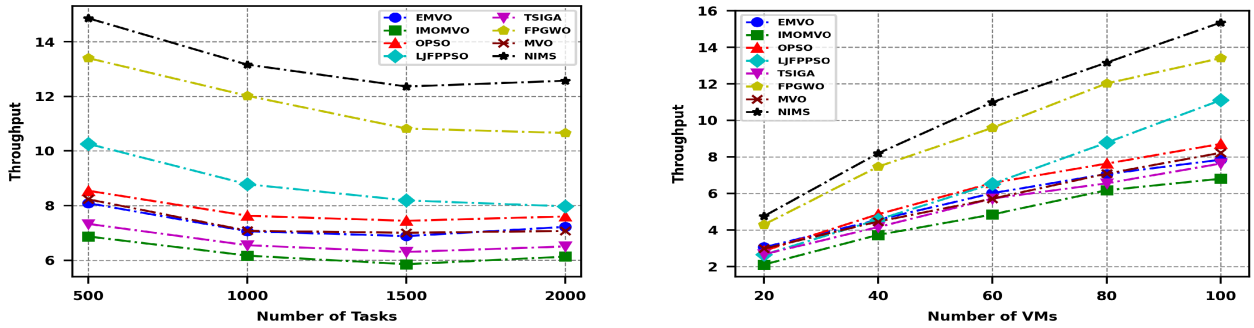

FIGURE 14. Throughput vs. Number of tasks and number of VMs using NASA-iPSC dataset.

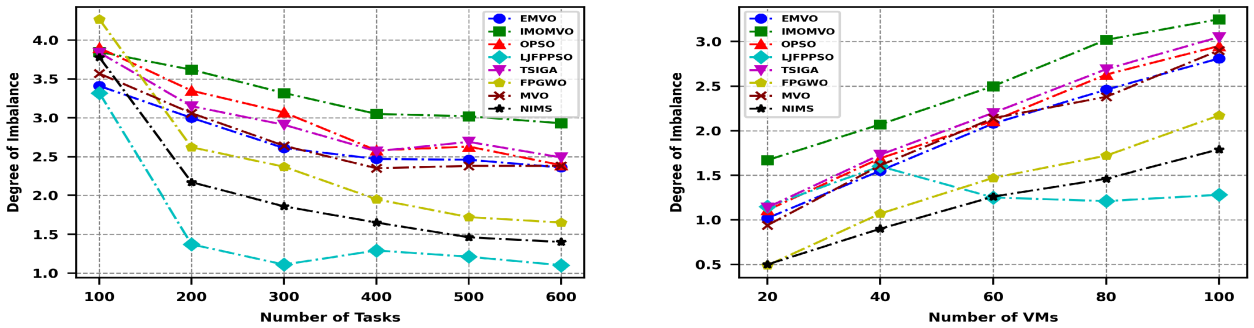

FIGURE 15. Degree of imbalance vs. Number of tasks and number of VMs using GoCJ dataset.

count), 65% over TSIGA, 26% over FPGWO, and 55% over MVO for the GoCJ dataset. For the HPC2N dataset, the improvement is observed up to 51% over EMVO, 73% over IMOMVO, 40% over OPSO, 69% over LJFPPSO, 58% over TSIGA, 7% over FPGWO, and 47% over MVO. Atlast, in case of NASA workload, the NIMS algorithm is able to

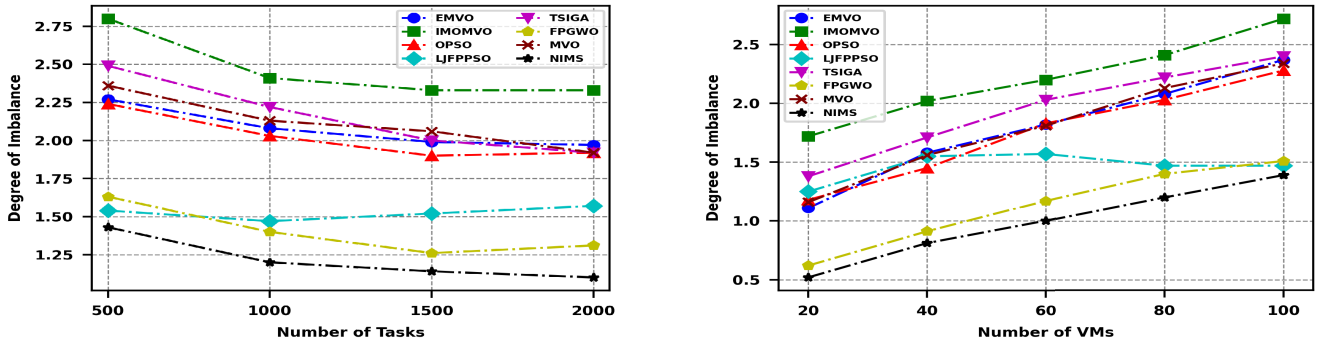

FIGURE 16. Degree of imbalance vs. Number of tasks and number of VMs using HPC2N dataset.

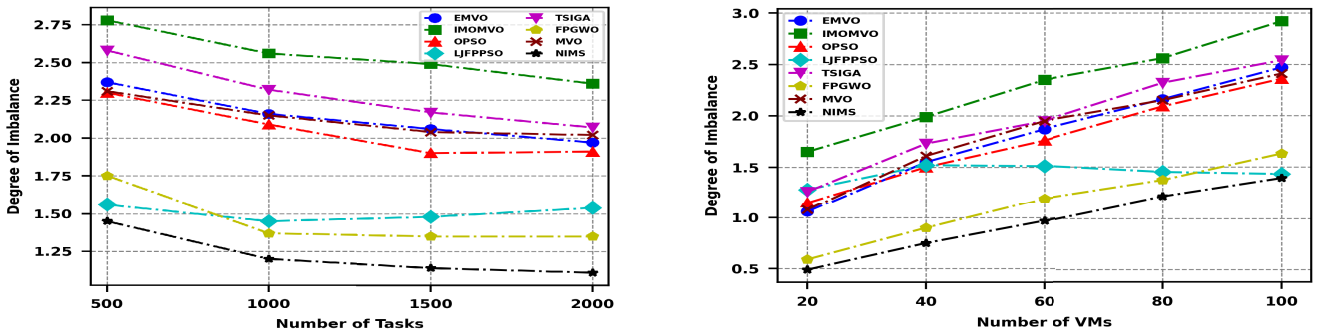

FIGURE 17. Degree of Imbalance vs. Number of tasks and number of VMs using NASA-iPSC dataset.

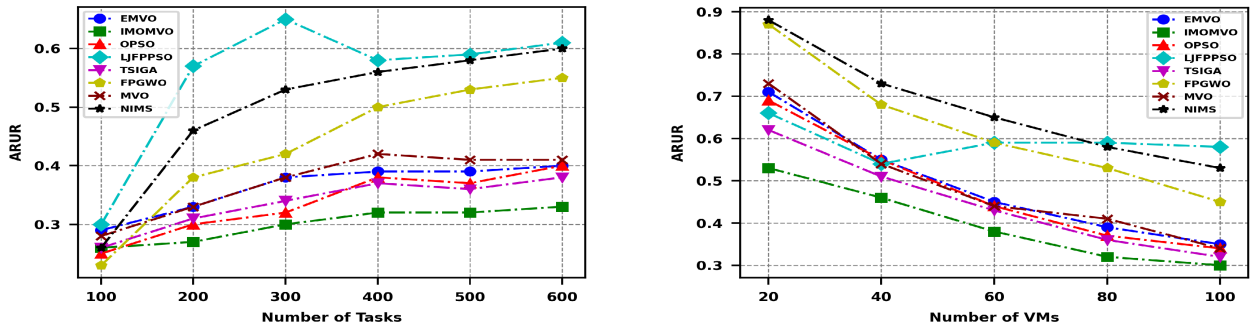

FIGURE 18. Average resource utilization rate vs. Number of tasks and number of VMs using GoCJ dataset.

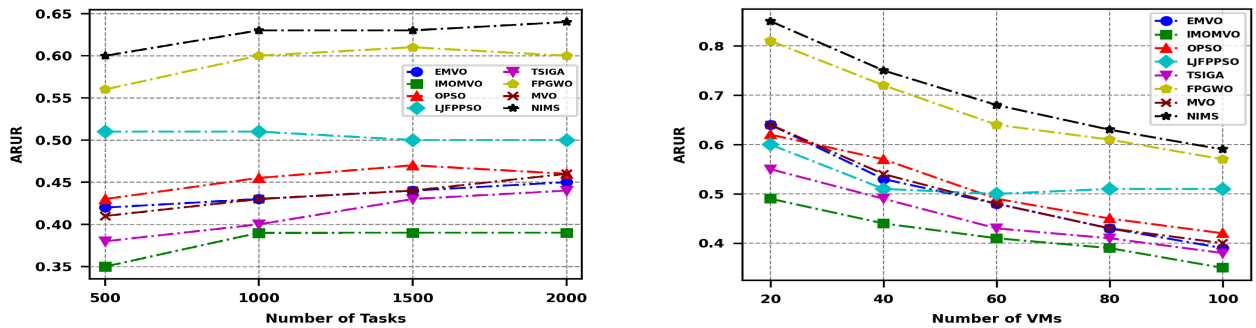

FIGURE 19. Average resource utilization rate vs. Number of tasks and number of VMs using HPC2N dataset.

improve by 55%, 78%, 47%, 47%, 62%, 11%, and 52% over EMVO, IMOMVO, OPSO, LJFPPSO, TSIGA, FPGWO, and MVO, respectively.

F. CONVERGENCE ANALYSIS

To evaluate the efficacy of the specified NIMS algorithm and its novel fitness-based neighborhood search technique,

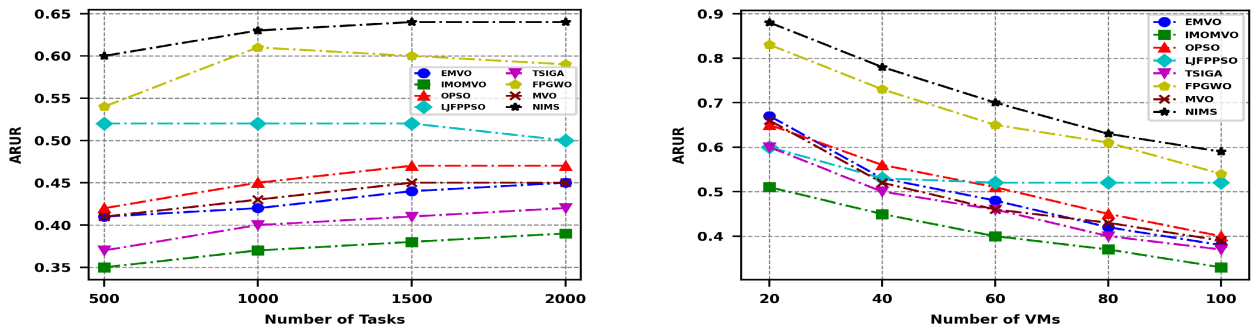

FIGURE 20. Average resource utilization rate vs. Number of tasks and number of VMs using NASA-iPSC dataset.

we tested the algorithm's convergence speed with selected comparative algorithms over all three real-world datasets considered in this study. The number of tasks is fixed at 500 for the GoCJ dataset and 1500 for the HPC2N and NASA-iPSC datasets, whereas the VM count is fixed at 50 in all three cases. Further, the convergence curve is plotted, taking the fitness function values produced by all the algorithms over 100 iterations. Figs. 21, 22, and 23 clearly display that all seven tested algorithms are optimizing the fitness values. However, our proposed NIMS algorithm shows better convergence behavior towards the optimal solution. Also, it is apparent from the plots that the suggested NIMS algorithm sooner converges to the lower fitness value in initial iterations itself against the comparative algorithms for all three workload datasets. This is due to the smart fitness-based update strategy adopted by the algorithm to improve the solution quality during iterations.

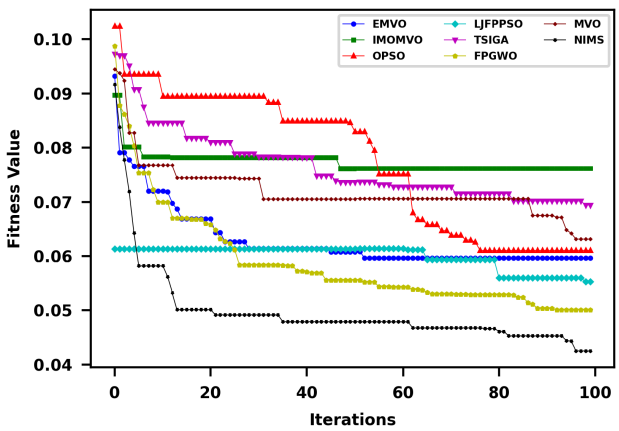

FIGURE 21. Convergence analysis using GoCJ workload.

TABLE 19. Test statistics $\chi^2(6)$ and p value observed by Kruskal Wallis test.

Algorithm	Count	Makespan		Energy	
		$\chi^2(6)$	P value	$\chi^2(6)$	P value
GoCJ	20	119.60	9.263e-23	119.54	9.521e-23
HPC2N	20	132.79	1.638e-25	132.93	1.527e-25
NASA	20	131.91	2.502e-25	131.98	2.418e-25

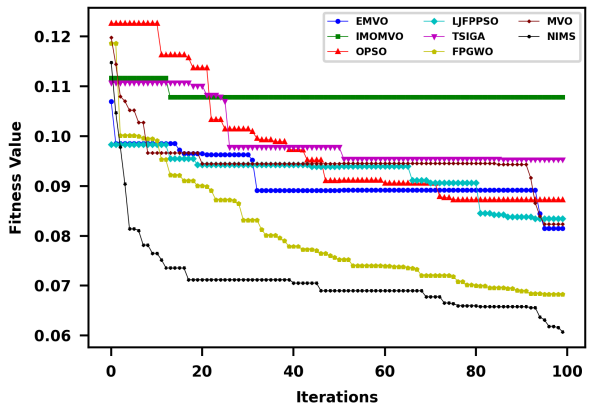

FIGURE 22. Convergence analysis using HPC2N workload.

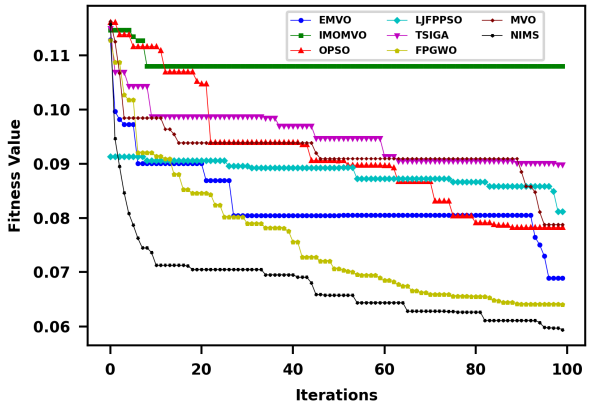

FIGURE 23. Convergence analysis using NASA workload.

TABLE 20. Statistical values for GoCJ dataset.

Algorithm	Makespan			Energy		
	Mean	SD	Var	Mean	SD	Var
EMVO	32.21	1.98	3.93	4.17	0.16	0.025
IMOMVO	45.11	3.54	12.56	5.19	0.28	0.078
OPSO	32.62	2.67	7.14	4.20	0.21	0.045
LJFPPSO	33.15	1.85	3.42	4.25	0.15	0.021
TSIGA	34.10	2.01	4.06	4.32	0.16	0.026
FPGWO	20.75	1.11	1.24	3.25	0.09	0.008
MVO	32.03	2.12	4.49	4.16	0.17	0.028
NIMS	20.1	0.61	0.37	3.20	0.05	0.002

VII. STATISTICAL ANALYSIS

To assess the performance variations between the proposed NIMS algorithm and other comparative algorithms for the

TABLE 21. Statistical values for HPC2N dataset.

Algorithm	Makespan			Energy		
	Mean	SD	Var	Mean	SD	Var
EMVO	23.89	1.47	2.18	3.04	0.12	0.013
IMOMVO	32.51	2.35	5.54	3.72	0.18	0.033
OPSO	22.56	1.64	2.71	2.94	0.13	0.017
LJFPPSO	25.89	1.38	1.91	3.20	0.11	0.012
TSIGA	25.05	1.52	2.31	3.13	0.11	0.013
FPGWO	14.99	0.93	0.87	2.33	0.07	0.005
MVO	23.08	1.52	2.31	2.98	0.12	0.014
NIMS	14.60	0.60	0.36	2.30	0.05	0.002

TABLE 22. Statistical values for NASA dataset.

Algorithm	Makespan			Energy		
	Mean	SD	Var	Mean	SD	Var
EMVO	1.19	0.07	0.005	0.15	0.006	3.1×10^{-5}
IMOMVO	1.61	0.06	0.004	0.18	0.005	2.6×10^{-5}
OPSO	1.18	0.07	0.006	0.15	0.006	3.8×10^{-5}
LJFPPSO	1.31	0.09	0.009	0.16	0.007	5.6×10^{-5}
TSIGA	1.28	0.07	0.005	0.16	0.006	3.4×10^{-5}
FPGWO	0.74	0.06	0.003	0.12	0.005	2.2×10^{-5}
MVO	1.17	0.058	0.003	0.15	0.004	2.1×10^{-5}
NIMS	0.73	0.02	0.0005	0.12	0.002	3×10^{-6}

task scheduling optimization problem, we performed a comprehensive statistical analysis using the Kruskal-Wallis test. This non-parametric test was chosen because it does not assume normality in the data, making it ideal for comparing multiple algorithms with potentially non-normally distributed performance metrics. The Kruskal-Wallis test is robust when comparing more than two independent groups. The null hypothesis (H0) assumes that the medians of each population are equal. Whereas the alternative hypothesis (H1) assumes that there is a significant difference in the population medians of one or the other groups. A p-value of less than 0.05 was considered statistically significant to reject the null hypothesis and indicate that at least one algorithm exhibits different performance. The analysis was carried out considering 500 tasks and 50 VMs on each dataset.Total 20 runs for each algorithm were included in the result to create a good sample size. The attained makespan and energy values are recorded for each run and considered for the test. Table 19 presents the test statistics $\chi^2(6)$ and p values attained by the test for all three dataset results. The higher $\chi^2(6)$ value indicates that there is a significant difference in the median performance values of each algorithm's performance. Similary, the p values attained for all three datasets are <0.05, indicating the rejection of the null hypothesis. In addition to the Kruskal-Wallis test, we also employed descriptive statistics, including mean, standard deviation, and variance, to capture performance variability across the algorithms. Tables 20 to 22 display the mean, standard deviation, and variance for makespan and energy values attained by each algorithm on GoCJ, HPC2N, and NASA datasets, respectively. It's clearly evident from the tables that our proposed NIMS algorithm achieved the lowest mean makespan and energy consumption with minimal variation across trials. Whereas, IMOMVO algorithms achieved the highest mean makespan and energy consumption values with significant deviation and variance. Also, the FPGWO algorithm has shown a mean makespan and energy consumption value close to the proposed NIMS algorithm; however, the deviation in values is found more in FPGWO. This significantly elaborates that the proposed NIMS algorithm has shown a consistent performance across multiple runs.

VIII. CONCLUSION, LIMITATIONS AND FUTURE RESEARCH DIRECTIONS

In this paper, we have developed an innovative task scheduling algorithm (NIMS) tailored for virtualized cloud environments. This algorithm harnesses the power of the Multiverse Optimizer (MVO) metaheuristic to achieve dual optimization of two critical metrics: makespan and energy consumption. By focusing on these metrics, NIMS not only reduces the time required for task completion but also minimizes overall energy consumption across virtual machines, aligning with the growing emphasis on green computing initiatives. The proposed algorithm integrates a novel fitness-based neighborhood search strategy that addresses the common challenge of premature convergence in metaheuristic optimization. This strategy significantly enhances the performance of the algorithm, leading to improved efficiency and sustainability in cloud computing. Finally performance of the NIMS algorithm is evaluated against seven state-of-the-art algorithms—EMVO, IMOMVO, OPSO, LJFPPSO, TSIGA, FPGWO, and MVO using widely recognized real-world benchmark datasets: GoCJ, HPC2N, and NASA. Our extensive simulations and experiments demonstrate that NIMS consistently outperforms these algorithms across numerous test cases. By effectively applying the novel fitness-based resource selection from the neighboring VMs, the proposed NIMS algorithm achieves a better task mapping solution.

Moreover, the results have proven that the proposed algorithm is also capable of avoiding local optima, thereby enhancing the global search capability of the MVO algorithm. This advancement ensures more reliable and efficient performance across various workload scenarios, contributing to the algorithm's adaptability and scalability in real-world cloud environments.

A. LIMITATIONS:

Despite its effectiveness, the NIMS algorithm has certain limitations:

1) The current version of NIMS is designed to operate only in static environments, limiting its applicability to dynamically changing scenarios.
2) NIMS is currently tailored to handle independent tasks and does not address task dependencies or interactions in its current form.
3) While the fitness-based neighborhood search strategy improves overall results, it does so at the cost of increased algorithmic time complexity in worst case.

B. FUTURE RESEARCH DIRECTIONS:

Future research will focus on exploring the following areas:

- Integration of Quality of Service (QoS) parameters and real-time workload variations to enhance the algorithm's applicability in more complex environments.
- Development of advanced algorithms to handle scenarios involving task dependencies and deadlines.
- Exploration of migration techniques to further decrease energy consumption.
- Investigation into the hybridization of the neighborhood search strategy with other optimization techniques for superior results.

REFERENCES

[1] Markets and Markets. (Dec. 2023). *Cloud Computing Market by Service Model (IaaS, PaaS, SaaS), Deployment Model(Public Cloud, Private Cloud, Hybrid Cloud), Organization Size, Vertical (BFSI, Telecommunications, Manufacturing, Retail & Customer Goods) and Region—Global Forecast to 2028*. TC 1228. Accessed: Apr. 12, 2024. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/cloud-computing-market-234.html

[2] S. G. Monserrate. (Feb. 2022). *The Staggering Ecological Impacts of Computation and the Cloud*. Accessed: Apr. 12, 2024. [Online]. Available: https://thereader.mitpress.mit.edu/the-staggering-ecological-impacts-of-computation-and-the-cloud/

[3] M. Farheen, S. Anand, and S. Sinha, "An efficient scheduling algorithm to improve the use of resources in cloud," *Int. J. Eng. Adv. Technol.*, vol. 8, no. 5, pp. 1761–1764, 2019.

[4] S. Choudhary, K. Suresh, and M. Sham, "Enhanced task scheduling mechanism in logistics based on improvised round Robin technique," in *Proc. IEEE 2nd Mysore Sub Sect. Int. Conf. (MysuruCon)*, Mysuru, India, Oct. 2022, pp. 1–7, doi: 10.1109/MysuruCon55714.2022.9972728.

[5] M. KumarB and A. S. Pillai, "Minimal task allocation in multiprocessors using improved next-fit for RM scheduling," *Int. J. Comput. Appl.*, vol. 47, no. 19, pp. 36–44, Jun. 2012, doi: 10.5120/7299-0546.

[6] S. Manishankar, R. Sandhya, and S. Bhagyashree, "Dynamic load balancing for cloud partition in public cloud model using VISTA scheduler algorithm," *J. Theor. Appl. Inf. Technol.*, vol. 87, no. 2, pp. 285–290, 2016.

[7] V. G. Polamuri, A. R. Poornima, and K. S. Sandeep, "Optimal hyperspectral band selection using robust multi-verse optimization algorithm," *Multimedia Tools Appl.*, vol. 82, no. 10, pp. 14663–14687, Apr. 2023, doi: 10.1007/s11042-022-13956-z.

[8] D. Mohan, S. Veni, and J. Aravinth, "Veridical discrimination of expurgated hyperspectral image utilizing multiverse optimization," in *Machine Learning and Computational Intelligence Techniques for Data Engineering* (Lecture Notes in Electrical Engineering), vol. 998, P. Singh, D. Singh, V. Tiwari, and S. Misra, Eds. Singapore: Springer, 2023, pp. 525–539, doi: 10.1007/978-981-99-0047-3_45.

[9] A. K. Abasi, A. T. Khader, M. A. Al-Betar, S. Naim, Z. A. A. Alyasseri, and S. N. Makhadmeh, "A novel hybrid multi-verse optimizer with K-means for text documents clustering," *Neural Comput. Appl.*, vol. 32, no. 23, pp. 17703–17729, Dec. 2020, doi: 10.1007/s00521-020-04945-0.

[10] X. Wang, D. Luo, X. Zhao, and Z. Sun, "Estimates of energy consumption in China using a self-adaptive multi-verse optimizer-based support vector machine with rolling cross-validation," *Energy*, vol. 152, pp. 539–548, Jun. 2018, doi: 10.1016/j.energy.2018.03.120.

[11] F. S. Prity, M. H. Gazi, and K. M. A. Uddin, "A review of task scheduling in cloud computing based on nature-inspired optimization algorithm," *Cluster Comput.*, vol. 26, no. 5, pp. 3037–3067, Oct. 2023, doi: 10.1007/s10586-023-04090-y.

[12] A. Nekooei-Joghdani and F. Safi-Esfahani, "Dynamic scheduling of independent tasks in cloud computing applying a new hybrid Metaheuristic algorithm including Gabor filter, opposition-based learning, multi-verse optimizer, and multi-tracker optimization algorithms," *J. Supercomput.*, vol. 78, no. 1, pp. 1182–1243, Jan. 2022, doi: 10.1007/s11227-021-03814-4.

[13] S. E. Shukri, R. Al-Sayyed, A. Hudaib, and S. Mirjalili, "Enhanced multi-verse optimizer for task scheduling in cloud computing environments," *Expert Syst. Appl.*, vol. 168, Apr. 2021, Art. no. 114230, doi: 10.1016/j.eswa.2020.114230.

[14] M. Otair, A. Alhmoud, H. Jia, M. Altalhi, A. M. Hussein, and L. Abualigah, "Optimized task scheduling in cloud computing using improved multi-verse optimizer," *Cluster Comput.*, vol. 25, no. 6, pp. 4221–4232, Dec. 2022, doi: 10.1007/s10586-022-03650-y.

[15] L. Abualigah and M. Alkhrabsheh, "Amended hybrid multi-verse optimizer with genetic algorithm for solving task scheduling problem in cloud computing," *J. Supercomput.*, vol. 78, no. 1, pp. 740–765, Jan. 2022, doi: 10.1007/s11227-021-03915-0.

[16] M. Agarwal and G. M. S. Srivastava, "Opposition-based learning inspired particle swarm optimization (OPSO) scheme for task scheduling problem in cloud computing," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 10, pp. 9855–9875, Oct. 2021, doi: 10.1007/s12652-020-02730-4.

[17] S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic initialization of PSO task scheduling algorithm in cloud computing," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 6, pp. 2370–2382, Jun. 2022, doi: 10.1016/j.jksuci.2020.11.002.

[18] K. Dubey and S. C. Sharma, "A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing," *Sustain. Comput., Informat. Syst.*, vol. 32, Dec. 2021, Art. no. 100605, doi: 10.1016/j.suscom.2021.100605.

[19] N. Panwar, S. Negi, M. M. S. Rauthan, and K. S. Vaisla, "TOPSIS–PSO inspired non-preemptive tasks scheduling algorithm in cloud environment," *Cluster Comput.*, vol. 22, no. 4, pp. 1379–1396, Dec. 2019, doi: 10.1007/s10586-019-02915-3.

[20] S. Al Shamaa, N. Harrabida, W. Shi, and M. St-Hilaire, "Particle swarm optimization with enhanced neighborhood search for task scheduling in cloud computing," in *Proc. IEEE Cloud Summit*, Fairfax, VA, USA, Oct. 2022, pp. 31–37, doi: 10.1109/CloudSummit54781.2022.00011.

[21] L. AlSuwaidan, S. Khan, R. Almakki, A. R. Baig, P. Sarkar, and A. E. S. Ahmed, "Swarm intelligence algorithms for optimal scheduling for cloud-based fuzzy systems," *Math. Problems Eng.*, vol. 2022, pp. 1–11, Jul. 2022, doi: 10.1155/2022/4255835.

[22] G. Saravanan, S. Neelakandan, P. Ezhumalai, and S. Maurya, "Improved wild horse optimization with levy flight algorithm for effective task scheduling in cloud computing," *J. Cloud Comput.*, vol. 12, no. 1, p. 24, Feb. 2023, doi: 10.1186/s13677-023-00401-1.

[23] P. Pabitha, K. Nivitha, C. Gunavathi, and B. Panjavarnam, "A chameleon and remora search optimization algorithm for handling task scheduling uncertainty problem in cloud computing," *Sustain. Comput., Informat. Syst.*, vol. 41, Jan. 2024, Art. no. 100944, doi: 10.1016/j.suscom.2023.100944.

[24] D. Alsadie, "TSMGWO: Optimizing task schedule using multi-objectives grey wolf optimizer for cloud data centers," *IEEE Access*, vol. 9, pp. 37707–37725, 2021, doi: 10.1109/ACCESS.2021.3063723.

[25] D. Alsadie, "A metaheuristic framework for dynamic virtual machine allocation with optimized task scheduling in cloud data centers," *IEEE Access*, vol. 9, pp. 74218–74233, 2021, doi: 10.1109/ACCESS.2021.3077901.

[26] K. Sreenu and M. Sreelatha, "W-scheduler: Whale optimization for task scheduling in cloud computing," *Cluster Comput.*, vol. 22, no. S1, pp. 1087–1098, Jan. 2019, doi: 10.1007/s10586-017-1055-5.

[27] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A WOA-based optimization approach for task scheduling in cloud computing systems," *IEEE Syst. J.*, vol. 14, no. 3, pp. 3117–3128, Sep. 2020, doi: 10.1109/JSYST.2019.2960088.

[28] P. Pirozmand, A. A. R. Hosseinabadi, M. Farrokhzad, M. Sadeghilalimi, S. Mirkamali, and A. Slowik, "Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing," *Neural Comput. Appl.*, vol. 33, no. 19, pp. 13075–13088, Oct. 2021, doi: 10.1007/s00521-021-06002-w.

[29] 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, doi: 10.1016/j.compeleceng.2021.107419.

[30] D. A. Amer, G. Attiya, and I. Ziedan, "An efficient multi-objective scheduling algorithm based on spider monkey and ant colony optimization in cloud computing," *Cluster Comput.*, vol. 27, no. 2, pp. 1799–1819, Apr. 2024, doi: 10.1007/s10586-023-04018-6.

[31] I. Attiya, M. Abd Elaziz, and S. Xiong, "Job scheduling in cloud computing using a modified Harris hawks optimization and simulated annealing algorithm," *Comput. Intell. Neurosci.*, vol. 2020, pp. 1–17, Mar. 2020, doi: 10.1155/2020/3504642.

[32] D. A. Abdulgader, A. Yousif, and A. Ali, "A discrete Prey–Predator algorithm for cloud task scheduling," *Appl. Sci.*, vol. 13, no. 20, p. 11447, Oct. 2023, doi: 10.3390/app132011447.

[33] S. Mangalampalli, S. K. Swain, and V. K. Mangalampalli, "Multi objective task scheduling in cloud computing using cat swarm optimization algorithm," *Arabian J. Sci. Eng.*, vol. 47, no. 2, pp. 1821–1830, Feb. 2022, doi: 10.1007/s13369-021-06076-7.

[34] I. Attiya, M. A. Elaziz, L. Abualigah, T. N. Nguyen, and A. A. A. El-Latif, "An improved hybrid swarm intelligence for scheduling IoT application tasks in the cloud," *IEEE Trans. Ind. Informat.*, vol. 18, no. 9, pp. 6264–6272, Sep. 2022, doi: 10.1109/TII.2022.3148288.

[35] I. Behera and S. Sobhanayak, "Task scheduling optimization in heterogeneous cloud computing environments: A hybrid GA-GWO approach," *J. Parallel Distrib. Comput.*, vol. 183, Jan. 2024, Art. no. 104766, doi: 10.1016/j.jpdc.2023.104766.

[36] A. N. Malti, M. Hakem, and B. Benmammar, "A new hybrid multi-objective optimization algorithm for task scheduling in cloud systems," *Cluster Comput.*, vol. 27, no. 3, pp. 2525–2548, Jul. 2023, doi: 10.1007/s10586-023-04099-3.

[37] 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, Jun. 2021, doi: 10.1007/s10462-020-09933-3.

[38] Y. Zhang and J. Wang, "Enhanced whale optimization algorithm for task scheduling in cloud computing environments," *J. Eng. Appl. Sci.*, vol. 71, no. 1, p. 121, May 2024, doi: 10.1186/s44147-024-00445-3.

[39] S. N. Hegde, D. B. Srinivas, M. A. Rajan, S. Rani, A. Kataria, and H. Min, "Multi-objective and multi constrained task scheduling framework for computational grids," *Sci. Rep.*, vol. 14, no. 1, p. 6521, Mar. 2024, doi: 10.1038/s41598-024-56957-8.

[40] R. Stewart, A. Raith, and O. Sinnen, "Optimising makespan and energy consumption in task scheduling for parallel systems," *Comput. Oper. Res.*, vol. 154, Jun. 2023, Art. no. 106212, doi: 10.1016/j.cor.2023.106212.

[41] S. Mirjalili, S. M. Mirjalili, and A. Hatamlou, "Multi-verse optimizer: A nature-inspired algorithm for global optimization," *Neural Comput. Appl.*, vol. 27, no. 2, pp. 495–513, Feb. 2016, doi: 10.1007/s00521-015-1870-7.

[42] L. Abualigah, "Multi-verse optimizer algorithm: A comprehensive survey of its results, variants, and applications," *Neural Comput. Appl.*, vol. 32, no. 16, pp. 12381–12401, Aug. 2020, doi: 10.1007/s00521-020-04839-1.

[43] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011, doi: 10.1002/spe.995.

[44] 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, Apr. 2021, doi: 10.1007/s10723-021-09548-0.

[45] 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, doi: 10.3390/data3040038.

[46] A. Hussain and M. Aleem, "GoCJ: Google cloud jobs dataset," Mendeley Data, Version 1, 2018, doi: 10.17632/b7bp6xhrcd.1.

[47] *Parallel Workloads Archive*. Accessed: Aug. 1, 2024. [Online]. Available: https://www.cs.huji.ac.il/labs/parallel/workload/logs.html

[48] S. Tiwari and B. M. Beena, "Energy cognizant scheduling in three-layer cloud architecture: A systematic review," in *Proc. 3rd Int. Conf. Issues Challenges Intell. Comput. Techn. (ICICT)*, Ghaziabad, India, Nov. 2022, pp. 1–6, doi: 10.1109/ICICT55121.2022.10064614.

[49] *SPEC Benchmark Results*. Accessed: Aug. 1, 2024. [Online]. Available: https://www.spec.org/power_ssj2008/results/res2008q4/

SHALINI TIWARI received the B.Tech. degree from Gautam Buddha Technical University, in 2010, and the M.Tech. degree from SRM University, Tamil Nadu, in 2013. She qualified for the Graduate Aptitude Test in Engineering (GATE), in 2011, and UGC-NET, in 2019. She is currently a full-time Researcher with the Department of Computer Science and Engineering, Amrita School of Computing, Amrita Vishwa Vidyapeetham, Bengaluru, India. Prior to this, she was an Assistant Professor with KIET Ghaziabad, from 2013 to 2014, followed by REVA University, Bengaluru, from 2015 to 2021. Her research interests include green computing, cloud computing, resource provisioning and allocation, algorithms, and operating systems.

BEENA B. M. (Senior Member, IEEE) received the B.E. degree in computer science and engineering from the PDA College of Engineering, Gulbarga University, the M.Tech. degree in computer science and engineering from the BMS College of Engineering, Bengaluru, and the Ph.D. degree from Visvesvaraya Technological University (VTU), Belagavi, Karnataka, India. She was with VTU affiliated engineering colleges in Bengaluru, and RUBIXE and Wells Fargo, Bengaluru, as a Senior Consultant, for five years. She is currently an Associate Professor with the Department of Computer Science and Engineering, Amrita School of Computing, Amrita Vishwa Vidyapeetham, Bengaluru, India. She has over 27 years of work experience both in industry and academics. She is also involved in industry funded projects in association with BOSCH, Bengaluru. She is guiding six Ph.D. students in green cloud computing and blockchain technology domain. Her research interests include green cloud computing, blockchain technology, and green artificial intelligence.