Concurrency and Computation: Practice and Experience

WILEY

RESEARCH ARTICLE

Multi-Objective Workflow Scheduling in Cloud Using Archimedes Optimization Algorithm

Shweta Kushwaha | Ravi Shankar Singh | Kanika Prajapati

Department of Computer Science and Engineering, Indian Institute of Technology (BHU), Varanasi, India

Correspondence: Shweta Kushwaha (shwetakushwaha.rs.cse21@iitbhu.ac.in)

Received: 7 May 2024 | **Revised:** 15 November 2024 | **Accepted:** 4 January 2025

Funding: The authors received no specific funding for this work.

Keywords: Archimedes optimization algorithm | cost-efficiency | local escaping operation | metaheuristics | optimization techniques | pareto-optimality | workflow scheduling

ABSTRACT

Cloud computing has changed the technology landscape for over a decade and led to an astounding growth in the number of applications it may be used for. Consequently, there has been a significant spike in the demand for improved algorithms to schedule workflows efficiently. These were mostly concerned with heuristic, metaheuristic, and hybrid approaches to workflow scheduling that mostly suffer from the problem of local optima entrapment. Due to such heavy traffic on the cloud resources, there is still a need for less computationally complex approaches. In light of this, this article proposes a novel approach: a multi-objective Modified Local Escaping Archimedes Optimization (MLEAO) algorithm for workflow scheduling. This strategy involves initialization of the population of Archimedes Optimization algorithm through the HEFT algorithm to provide an inclination towards the solutions with improved makespan while achieving a cost-efficient workflow scheduling decision and avoiding the problem of local optima entrapment using a local escaping operation. To validate the efficacy of our approach, we conducted extensive experiments using scientific workflows as benchmarks. Through our investigations, we significantly improved makespan, cost, resource utilization, and energy consumption. Moreover, the effectiveness of our proposed approach is also verified by performance metrics such as hypervolume, s-metric, and dominance relationships between the proposed and state-of-the-art approaches.

1 | Introduction

Cloud computing has significantly transformed various sectors over the past decade, offering scalable computational and storage resources at reduced costs. This has facilitated the growth of data volumes and the complexity of processing demands, driven largely by the proliferation of Internet of Things (IoT) devices across industries such as business, education, healthcare, and agriculture. To meet the increasing demand for efficient processing, there is a growing need for scalable cloud infrastructure, alongside automation and continuous upgrades to hardware and software systems. Achieving optimal performance in cloud environments hinges on efficient resource allocation and management, which remain crucial to ensuring system reliability and performance [1].

In cloud platforms, workflows—comprising independent or interdependent tasks—are scheduled and executed by allocating resources to individual tasks and determining their execution order while accounting for dependencies. Workflow scheduling is a well-known NP-hard problem, characterized by its complexity and computational challenges [2]. As cloud computing

Abbreviations: GA, genetic algorithm; PSO, particle swarm optimization; VM, virtual machines.

applications evolve, the need for effective scheduling algorithms becomes increasingly important. These algorithms must balance multiple objectives, such as resource utilization, energy efficiency, and execution time, while avoiding pitfalls such as local optima entrapment, where algorithms fail to find the global optimum solution due to their reliance on local search heuristics.

Although many heuristic and metaheuristic approaches have been developed to tackle workflow scheduling, they often suffer from the problem of getting trapped in local optima. For example, various heuristics—such as enhanced round-robin and shared-file-aware algorithms—have shown improvements in specific areas but often fail to avoid local optima, limiting their scalability and adaptability [3, 4]. Similarly, more complex metaheuristic approaches, including ant colony optimization and genetic algorithms, have been proposed to optimize execution time and costs, yet they remain vulnerable to local optima entrapment, particularly when handling large, dynamic workflows [5, 6]. Even hybrid methods, which combine multiple algorithms to improve performance, struggle with local optima issues [7, 8]. The persistence of local optima entrapment in existing scheduling methods calls for innovative strategies that enhance exploration capabilities to find better, globally optimal solutions.

This paper addresses the challenge of local optima entrapment in workflow scheduling by introducing a novel metaheuristic approach based on the Archimedes principle, named the Archimedes Optimization Algorithm (AOA). The AOA draws inspiration from the physical principle of buoyancy, which seeks equilibrium between objects and fluid, to iteratively refine potential solutions in the scheduling process. By employing a method that explores multiple candidate solutions in parallel and adjusting their positions based on a balance of forces, the algorithm aims to provide better exploration and exploitation capabilities to avoid local optima traps and converge to more globally optimal solutions. To further avoid the local optima entrapment problem, the local escaping operation has been implemented in addition to HEFT and Archimedes Optimization algorithm.

The key contributions of this paper are:

- A new multi-objective metaheuristic approach, Modified Local Escaping Archimedes Optimization (MLEAO) Algorithm, focuses on improving workflow scheduling while addressing local optima entrapment, makespan, and cost optimization.
- Initialization using the HEFT algorithm to provide an inclination towards the solutions with better makespan values.
- A fitness function that integrates multiple objectives, including makespan and cost, prioritizing solutions that meet both improved makespan and cost values.
- Evaluation and validation of the algorithm through experimental results using well-known scientific workflows, comparing its performance against existing state-of-the-art methods based on metrics such as makespan, cost, energy consumption, and resource utilization.
- A detailed analysis of the algorithm's ability to avoid local optima traps through various Pareto optimality metrics, such as hypervolume, dominance, and s-metric.

Comprehensive experiments reveal that the proposed MLEAO algorithm achieves substantial improvements in workflow scheduling, enhancing critical metrics such as makespan, cost, energy efficiency, and resource utilization. These findings confirm MLEAO's effectiveness in navigating local optima issues and delivering more resilient, globally optimal scheduling outcomes.

1.1 | Structure of the Paper

The rest of the paper is organized as follows: Section 2 presents the related works. Section 3 describes the workflow scheduling problem in the cloud environment. Section 4 presents the proposed approach in detail, the background overview of Archimedes optimization algorithm and the local escaping operation and further the Modified Local Escaping Archimedes Optimization algorithm. Section 5 presents the details of the experiments performed and the results achieved. Section 6 presents the conclusions and future research directions.

2 | Related Work

Heuristics offer efficient problem-solving strategies, especially valuable for time-intensive tasks. Numerous heuristic techniques have been introduced for workflow scheduling. Table 1 presents the overview of various approaches proposed for workflow scheduling. Sanaj and Prathap proposed an enhanced round-robin algorithm using time slices to improve task processing and reduce wait times, but it only applies to independent tasks and does not address local optima traps [3]. Tuli et al. introduced a shared-file-aware algorithm for resource provisioning to reduce execution time, but it only supports a bag of tasks model and ignores local optima issues [4]. Ma et al. presented a heuristic for real-time multi-workflow scheduling in cloud environments, optimizing resource utilization and scheduling time, but it struggles with local optima [9]. Tang et al. developed a dynamic task scheduling technique focused on cost optimization and energy efficiency, but its heuristic nature results in local optima challenges [26]. Medara et al. proposed an energy-efficient scheduling technique using task ranking, clustering, and slack algorithms to reduce energy consumption, but it is prone to local optima, especially under strict energy constraints [10]. Medara, Singh and Sompalli also introduced a HEFT-based scheduling algorithm for energy and cost optimization, yet it risks local optima due to its heuristic design [11]. Taghinezhad-Niar, Pashazadeh and Taheri also suggested an energy-efficient, budget-driven scheduling method for both DVFS and non-DVFS resources, but it is still susceptible to local optima due to a narrow problem space [12].

Heuristic methods, while often expedient for solving NP-complete problems, tend to be limited by local optima traps. To address this, metaheuristic algorithms have been introduced to enhance exploration capabilities through multiple initial solutions. Pham and Fahringer proposed an evolutionary multi-objective scheduling approach for cloud resource optimization, improving makespan and resource volatility management, yet it is prone to local optima due to load and system utilization balance issues [13]. Medara et al. introduced a discrete water wave optimization-based energy-aware approach for VM migration, improving resource allocation, but it remains susceptible to local optima due to limited adaptability [10]. Nabi and Ahmed

TABLE 1 | Advantages and limitations of state-of-the-art approaches.

Article	Key idea	Advantages	Limitations
[3]	An enhanced round-robin algorithm using time slices	Improves task processing and reduces wait times	Only applies to independent tasks & doesn't address local optima traps
[4]	A shared-file-aware algorithm for resource provisioning	Reduces execution time	Only supports a bag of tasks model and ignores local optima issues
[9]	A heuristic for real-time multi-workflow scheduling	Optimizes resource utilization and scheduling time	Struggles with local optima
[10]	An energy-efficient scheduling technique using task ranking, clustering, and slack algorithms	Reduces energy consumption	Prone to local optima
[11]	A HEFT-based scheduling algorithm	Energy and cost optimization	Risks local optima due to its heuristic design
[12]	Scheduling method for both DVFS and non-DVFS resources	Energy-efficient and budget-driven	Susceptible to local optima due to a narrow problem space
[13]	An evolutionary multi-objective scheduling approach for cloud resource optimization	Improves makespan and resource management	Load and system utilization balance issues
[14]	A discrete water wave optimization-based energy-aware approach for VM migration	Improves resource allocation	Limited adaptability
[15]	An energy-efficient scheduling technique using threshold-based VM task consolidation	Energy-efficient	Lack of consideration for VM behavior fluctuations
[16]	Grey wolf optimization-based scheduling	Improves cost, response time, and load balancing	Limited adaptability for workflow variations
[17]	A harmony-inspired genetic algorithm to balance exploration and exploitation	Reduces energy and makespan	Faces local optima issues
[18]	A multi-verse optimizer for task scheduling	Improves resource usage	Lack of energy cost consideration
[19]	An enhanced task type-first algorithm with hibernation features	Reduces costs	Limited by a small search space
[20]	A hybrid swarm intelligence approach combining manta ray foraging and salp swarm algorithms	Improves throughput and makespan	Vulnerable to local optima traps
[21]	A hybrid cloud task scheduling method using genetic and gravitational search algorithms	Enhances VM performance and energy calculation	Faces local optima issues due to its fixed cost-based optimization focus
[22]	A hybrid PCP–ACO algorithm	Minimizes workflow costs while meeting deadlines	Susceptible to local optima entrapment
[23]	Enhanced artificial rabbits optimization (ARO) for task scheduling	Improves exploration and service time	Faces local optima challenges
[24]	A multi-objective approach using modified NCCLA	Improves exploration-exploitation balance	Vulnerable to local optima
[25]	Multi-objective scheduling approach for scientific workflows using Adaptive Dingo Optimization	Minimizes duration and cost	Local optima entrapment

modified particle swarm optimization to optimize makespan, cost, and resource utilization, but the design does not fully avoid local optima due to its isolated task scheduling approach [27]. Singh and Kumar proposed an energy-efficient scheduling technique using threshold-based VM task consolidation, but it is vulnerable to local optima due to the lack of consideration for VM behavior fluctuations [15]. Sefati, Mousavinasab and Zareh Farkhady introduced grey wolf optimization-based scheduling to improve cost, response time, and load balancing, but it struggles with local optima due to limited adaptability for workflow variations [16].

Hybrid methods combining algorithms address these challenges by leveraging multi-algorithm strengths. Sharma and Garg introduced a harmony-inspired genetic algorithm to balance exploration and exploitation, reducing energy and makespan, but it still faces local optima issues [17]. Shukri et al. proposed a multi-verse optimizer for task scheduling, improving resource use but remaining prone to local optima and the lack of energy

cost consideration [18]. Sun et al. introduced an enhanced task type-first algorithm with hibernation features to reduce costs, but it is limited by a small search space, making it prone to local optima [19]. Attiya et al. proposed a hybrid swarm intelligence approach combining manta ray foraging and salp swarm algorithms to improve throughput and makespan, but it is vulnerable to local optima traps and only applicable for independent tasks [20]. Pirozmand et al. introduced a hybrid cloud task scheduling method using genetic and gravitational search algorithms to enhance VM performance and energy consumption, but it still faces local optima issues due to its fixed cost-based optimization focus [21]. Shobeiri et al. proposed a hybrid PCP–ACO algorithm to minimize workflow costs while meeting deadlines, but it remains susceptible to local optima entrapment [22]. Ghafari and Mansouri enhanced artificial rabbits optimization (ARO) for task scheduling in fog computing, improving exploration and service time, but it still faces local optima challenges [23]. Zade, Javidi and Mansouri introduced a multi-objective approach using modified NCCLA to improve exploration-exploitation balance, but it remains vulnerable to local optima despite improvements in VM structure [24]. Mary proposed a multi-objective scheduling approach for scientific workflows using Adaptive Dingo Optimization (ADO) to minimize duration and cost, but local optima entrapment remains a limitation [25].

Although numerous heuristic, metaheuristic, and hybrid approaches have been proposed, most struggle with local optima challenges. There's still a need for multi-objective approaches with lesser computational complexity that can schedule workflows optimally can also avoid the local optima entrapment problem. The Archimedes Optimization algorithm, known for its effective balance between exploration and exploitation phases, offers a potential solution with reduced complexity compared to traditional metaheuristics and hybrid methods. In this work, we have proposed a modified metaheuristic approach, Modified Local Escaping Archimedes Optimization (MLEAO) Algorithm, to solve the workflow scheduling problem using the HEFT algorithm, Archimedes Optimization algorithm, and Local Escaping Operation to address local optima challenges in workflow scheduling, enhancing performance across key objectives like makespan and cost.

3 | Workflow Scheduling Model & Problem Formulation

3.1 | Workflow Model

A workflow can be explained as a directed acyclic graph $G(Tsk, E)$ where Tsk is a set of T tasks, $Tsk = \{t_1, t_2, t_3, \ldots t_T\}$, and E is the set of directed edges between the tasks. An edge from the task t_h to t_i can be denoted by $e_{h,i}$. This notation signifies the precedence relationship of the tasks. According to this notation, task t_i can start its execution only if task t_h has finished its execution and all the input data required for the execution of t_i is received. Here, task t_h is the parent task and task t_i is the child task of task t_h. Since workflows process data in files, let the input data file list be $IDFL_i$ and the output data file list be $ODFL_i$ for task t_i. Let the task length of t_i be TL_i. Hence, the workflow model can be defined as $Wf = \{G, \{TL_i, IDFL_i, ODFL_i\}\}$, which signifies that a workflow is a graph of tasks, where each task has a task length, an input data file, an output data file and a set of edges that denotes the precedence order of the tasks according to which these data files will be transferred. Figure 1 presents the basic structure of the scientific workflows we have used as the benchmark for scheduling in this work.

3.2 | Cloud Resource Model

Workflows submitted to cloud computing environments are generally executed on virtual machines. These virtual machines are deployed on the hosts in the cloud computing environment, and the host allocates the computing power and bandwidth of the virtual machine. The number of virtual machines in a cloud environment is usually large and considered infinite. Still, the type of the virtual machines is finite. Based on the types of virtual machines, the capacities of the VMs vary. Following are some basic definitions to explain these variations. Let us consider a datacenter D having a set of V active virtual machines, $Vm = \{v_1, v_2, v_3, \ldots v_V\}$. The notations of the symbols used are explained in the Table 2.

Virtual Machine Instance Capacity: The capacity of any virtual machine instance refers to the total number of instructions that it can process in one second. It can be given by the product of the number of processing elements and the size of each processing element (in MIPS) as given in Equation (1).

$$C_{v_j} = PE^n_{v_j} \times PE^s_{v_j} \tag{1}$$

Capacity of Multiple Virtual Machine Instances: To compute the capacity of multiple virtual machine instances in a datacenter D, we can add the capacities of individual virtual machine instances active on the datacenter D as given in Equation (2). Let V be the total number of virtual machines active on datacenter D, then

$$C_D = \sum_{v=1}^{V} C_{v_j} \tag{2}$$

The total capacity of multiple virtual machine instances active on a datacenter must not exceed the datacenter's capacity as given in

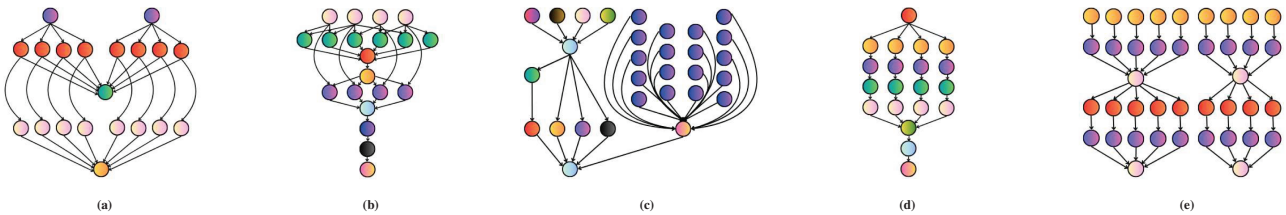

FIGURE 1 | Structure of scientific workflows. (a) CyberShake. (b) Montage. (c) SIPHT. (d) Epigenomics. (e) Inspiral.

TABLE 2 | Symbols and their notations.

Symbol	Notation	Symbol	Notation
T	Number of tasks	Vm	Set of Virtual machines active on a datacenter
Tsk	Set of T tasks	C_{v_j}	Capacity of jth virtual machine
t_i	ith task	$PE^n_{v_j}$	Number of processing elements of jth virtual machine
V	Number of virtual machines	C_{Vm}	Total capacity of the virtual machines in set Vm
v_j	jth virtual machine	$PE^s_{v_j}$	Size of processing elements of jth virtual machine (in MIPS)
C_D	Capacity of Datacenter	$ET^{v_j}_{t_i}$	Execution time of task t_i on VM v_j
t^s_i	Size of task t_i	$TT^{v_k}_{v_j}$	Transmission time from VM v_j to VM v_k
Pr_{t_i}	Predecessors of task t_i	$DS^{v_k}_{v_j}$	Size of data to be transferred from VM v_j to VM v_k
WT_{t_i}	Waiting time of task t_i	$\overline{B}$	Average bandwidth of Datacenter
CT_{t_i}	Completion time of task t_i	$DCT^{Pr_{t_i}}_{t_i}$	Communication time from predecessor tasks of t_i to task t_i
C_{v_j}	Cost of VM v_j	M_{Wf}	Makespan for the execution of workflow Wf
C_B	Cost of Bandwidth	$EC^{v_j}_{t_i}$	Execution cost for task t_i on VM v_j
V^r_j	Running voltage of VM v_j	EC_{Wf}	Execution cost for the workflow Wf
V^M_j	Maximum voltage of VM v_j	$TC^{t_i}_{v_k}$	Transfer cost to execute task t_i on VM v_k
V^m_j	Minimum voltage of VM v_j	TC_{Wf}	Transfer cost for the workflow
F^r_j	Running frequency of VM v_j	$Cost_{t_i}$	Total Cost for execution of task t_i
F^M_j	Maximum frequency of VM v_j	$Cost_{Wf}$	Total Cost for execution of Workflow Wf
F^m_j	Minimum frequency of VM v_j	E_{Wf}	Energy Consumption for execution of workflow Wf
P_j	Power consumption of VM v_j	U_{Wf}	Resource Utilization for execution of workflow Wf
AT_j	Active time of VM v_j	N	Number of objects in population
m	Previous Iteration	M	Maximum number of iterations
$m+1$	Current iteration	O_b	Object with best fitness value
ϑ	A random number	u	Upper normalization range
b_u	Upper bound	l	Lower normalization range
b_l	Lower bound	TF^{m+1}	Transfer factor of current iteration
O_n	nth $Object$	DF^{m+1}	Density factor of current iteration
ρ_n	Position vector of O_n	θ	Threshold to decide exploration and exploitation phase
D_n	Density vector of O_n	D^{m+1}_n	Density vector of O_n in $(m+1)$th iteration
V_n	Volume vector of O_n	V^{m+1}_n	Volume vector of O_n in $(m+1)$th iteration
A_n	Acceleration vector of O_n	A^{m+1}_n	Acceleration vector of O_n in $(m+1)$th iteration
F_n	Fitness of O_n	NA^{m+1}_n	Normalized Acceleration vector of O_n in $(m+1)$th iteration
O_ϑ	Randomly chosen Object	ρ^{m+1}_n	Position vector of O_n in $(m+1)$th iteration
α	Impact constant	F	Flag to change the direction of motion of Object

Equation (3). Datacenter capacity is the total capacity of virtual machines in a datacenter D.

$$C_{Vm} \leq C_D \tag{3}$$

3.3 | Workflow Scheduling Model

Scheduling of workflows in the cloud environment may be done in two types: static and dynamic scheduling. Static scheduling requires all the information of the workflow and the resources at the time of compiling the scheduling algorithm. In dynamic scheduling, all this information is known at the run time of the scheduling algorithm. Unlike dynamic scheduling, static scheduling receives all the information at a prior stage. It searches globally in the solution space, leading to better results than the dynamic scheduling methods. Hence, this work focuses on the static scheduling approach. The following are some basic definitions of workflow scheduling.

Execution Time: The execution time of a task t_i on a virtual machine v_j is the time required for the VM to complete the task. It depends on the task length TL_i (the total number of instructions) and the VM's capacity C_{v_j} (the instructions it can process per second) [28]. The execution time is calculated as:

$$ET_{i,j} = \frac{TL_i}{C_{v_j}} \tag{4}$$

This measures how long the task will take based on the VM's processing power.

Transfer Time: The transfer time is the time required to transfer data or files from the parent task to the child task when they are not executed on the same virtual machine. It depends on the size of the data being transferred and the bandwidth between the two VMs [28]. The transfer time is given by:

$$TT_{v_j}^{v_k} = \frac{DS_{v_j}^{v_k}}{\overline{B}} \tag{5}$$

Here $DS_{v_j}^{v_k}$ is the data size being transferred from VM v_j to VM v_k, and $\overline{B}$ is the average bandwidth.

Data Communication Time: The data communication time is the maximum time needed to transfer data from all the virtual machines executing the predecessor tasks to the virtual machine where the task t_i will be executed [28]. If task t_i is executed on VM v_j, and its predecessor tasks Pr_{t_i} are executed on a set of VMs v_x, the data communication time is given by:

$$DCT_{t_i}^{Pr_{t_i}} = \max(TT_{v_x}^{v_j}) \tag{6}$$

This represents the longest transfer time from any predecessor VM to the current VM.

Waiting Time: The waiting time WT_{t_i} of a task t_i is defined as the maximum of the completion times of all its predecessor tasks Pr_{t_i} in the workflow Wf [28]. If task t_i is executed on a different virtual machine than its predecessors, the data communication time is added to the waiting time. The waiting time can be expressed mathematically as follows:

$$WT_{t_i} = \begin{cases} 0 & \text{if } Pr_{t_i} = \emptyset, \\ \max(CT_{Pr_{t_i}}) & \text{if } Pr_{t_i} \neq \emptyset \text{ and } v^{t_i} = v^{Pr_{t_i}} \\ \max(CT_{Pr_{t_i}}) + DCT_{t_i}^{Pr_{t_i}} & \text{if } Pr_{t_i} \neq \emptyset \text{ and } v^{t_i} \neq v^{Pr_{t_i}} \end{cases} \tag{7}$$

Here, v^{t_i} denotes the virtual machine on which task t_i has to be executed, and ϕ denotes the NULL value. This defines how waiting time is calculated based on the presence of predecessor tasks and whether the current task and its predecessors share the same VM.

Completion Time: The completion time CT_{t_i} of a task t_i in a workflow Wf is the moment at which the task finishes executing on the virtual machine instance v_j [28]. It is determined based on whether the task has predecessor tasks. The completion time can be formulated as follows:

$$CT_{t_i} = \begin{cases} ET_{t_i}^{v_j}, & \text{if } Pr_{t_i} = \emptyset \\ WT_{t_i} + ET_{t_i}^{v_j}, & \text{if } Pr_{t_i} \neq \emptyset \end{cases} \tag{8}$$

Here, the completion time is equal to the execution time if there are no predecessor tasks, or the waiting time plus the execution time if there are predecessor tasks.

Makespan: The makespan is defined as the maximum completion time among all the tasks in a workflow. It indicates the total time required to complete the workflow, encompassing all tasks [28]. The makespan can be expressed mathematically as:

$$M_{Wf} = \max(CT_{t_i}), \quad \text{for distinct } 1 \leq i \leq T \tag{9}$$

3.3.1 | Cost Model

Execution Cost: The execution cost $EC_{t_i}^{v_j}$ is the cost associated with executing a task t_i on a virtual machine instance v_j [28]. It is calculated using the formula:

$$EC_{t_i}^{v_j} = ET_{t_i}^{v_j} \times C_{v_j} \tag{10}$$

In this equation, $ET_{t_i}^{v_j}$ represents the execution time of the task on the VM, and C_{v_j} is the cost of using that VM, expressed in dollars per hour. The total execution cost of the workflow can be computed as the sum of the execution costs of all individual tasks:

$$EC_{Wf} = \sum_{i=1}^{T} EC_{t_i}^{v_j} \tag{11}$$

Transfer Cost: The transfer cost $TC_{t_i}^{v_k}$ is the cost incurred when transferring data or files required by a child task t_i from its parent tasks, specifically when the child task is executed on a different virtual machine v_k than its parents [28]. This cost is calculated using the formula:

$$TC_{t_i}^{v_k} = \frac{DS_{v_j}^{v_k} \times C_B}{\overline{B}} \tag{12}$$

In this equation, $DS_{v_j}^{v_k}$ is the data size transferred from the parent VM v_j to the child VM v_k, C_B represents the cost per unit of data transferred and $\overline{B}$ is the average bandwidth for the data transfer. Further, the total transfer cost for the entire workflow execution can be calculated as:

$$TC_{Wf} = \sum_{i=1}^{T} TC_{t_i}^{v_k} \tag{13}$$

Total Cost: The total cost of a workflow execution is the combined sum of the execution cost and the transfer cost [28]. Specifically, the total cost incurred for executing a task t_i on a virtual machine instance v_j can be expressed as:

$$Cost_{t_i} = TC_{t_i}^{v_j} + EC_{t_i}^{v_j} \tag{14}$$

In this equation, $TC_{t_i}^{v_j}$ is the transfer cost associated with the task, and $EC_{t_i}^{v_j}$ is the execution cost for that task on the VM. The overall total cost of executing the entire workflow is given by the equation:

$$Cost_{Wf} = EC_{Wf} + TC_{Wf} \tag{15}$$

Here, EC_{Wf} is the total execution cost of the workflow, and TC_{Wf} is the total transfer cost.

3.3.2 | Energy Model

Running Voltage: The running voltage V_j^r of a virtual machine instance v_j refers to the voltage level at which that instance

operates. It is determined based on the minimum and maximum voltages, as well as the current frequency of the virtual machine [10]. The running voltage can be expressed mathematically as:

$$V_j^r = V_j^m + (V_j^M - V_j^m) \times \frac{F_j^r - F_j^m}{F_j^M - F_j^m} \tag{16}$$

In this equation, V_j^m is the minimum voltage, V_j^M is the maximum voltage, F_j^r is the current frequency, F_j^m is the minimum frequency and F_j^M is the maximum frequency. This formula shows how the running voltage varies with frequency within the specified voltage range.

Power Consumption: The power consumption P_j of a virtual machine instance is defined as the product of the running frequency F_j^r and the square of its running voltage V_v^r [10]. This relationship can be expressed mathematically as:

$$P_j = F_j^r \times V_v^{r2} \tag{17}$$

In this equation, the power is measured in watts (W).

Energy Consumption: It refers to the total energy consumed by the datacenter during the execution of a workflow, measured in watt-hours (Wh) [10]. It is calculated by summing the power consumption of all virtual machines over their active time AT_j:

$$E_{Wf} = \sum_{j=1}^{V} P_j \times AT_j \tag{18}$$

This equation captures the overall energy usage of the datacenter for the workflow execution.

3.3.3 | Utilization Model

Utilization: The resource utilization $U_{Wf(\%)}$ for executing a workflow Wf on a datacenter D with V active virtual machines (VMs) quantifies how efficiently the available resources are used. It is calculated as the ratio of the total active time of all VMs AT_j to the makespan M_{Wf}, normalized by the number of active VMs, and expressed as a percentage [28]. The formula is given by:

$$U_{Wf(\%)} = \left(\frac{\sum_{j=1}^{V} \frac{AT_j}{M_{Wf}}}{V} \right) \times 100 \tag{19}$$

This equation helps to assess the effectiveness of resource utilization during the workflow execution.

3.4 | Problem Formulation

This work aims to address the multi-objective workflow scheduling problem, which seeks to find a compromised solution that optimizes both the makespan M_{Wf} and the total cost $Cost_{Wf}$ of executing a workflow. In this context, a "solution" refers to the resource provisioning sequence that adheres to the dependency constraints of all tasks within the workflow. The optimization problem can be mathematically formulated as follows:

$$\{\text{minimize}(M_{Wf} \text{ and } Cost_{Wf})\} \tag{20}$$

This formulation highlights the dual focus of the scheduling strategy on minimizing both time and financial expenditure during workflow execution.

4 | The Proposed Approach

This section presents a comprehensive explanation of the proposed Modified Local Escaping Archimedes Optimization Algorithm (MLEAO), designed to address the problem outlined in the preceding section. The primary motivation for enhancing the original Archimedes Optimization Algorithm (AOA) stems from the need to improve its search capabilities and address inherent limitations, particularly when applied to complex real-world optimization problems. The MLEAO integrates two key strategies: first, the initialization process incorporates a solution seeded from the Heterogeneous Earliest Finish Time (HEFT) algorithm, and second, the Local Escaping Operation (LEO) is employed to mitigate the risk of premature convergence and avoid entrapment in local optima. Following the HEFT-based initialization, the algorithm proceeds through the conventional steps of the AOA, after which the LEO strategy is applied to further enhance its overall performance.

4.1 | Background Overview

This subsection provides a detailed account of the methods integrated into the proposed approach. It elaborates on the modified version of the Archimedes Optimization Algorithm (AOA), which incorporates the Heterogeneous Earliest Finish Time (HEFT) solution as one of the initial candidate solutions. Additionally, it thoroughly describes the application of the Local Escaping Operation (LEO), which is designed to enhance the algorithm's ability to evade local optima and improve overall solution quality.

4.1.1 | Archimedes Optimization Algorithm

Archimedes Optimization algorithm is a population-based metaheuristic algorithm proposed in 2021 by Hashim et al. [29]. This algorithm is a physics-based metaheuristic algorithm inspired by Archimedes' principle. It explains the force's behavior when an object is submerged entirely or partially in a fluid. Figure 2 demonstrates the Archimedes principle. At the beginning of the algorithm, collisions between the objects occur, leading to a change in position due to the updated acceleration resulting from collision. During this time, the algorithm tries to get a solution in the global solution space so that it doesn't get stuck in the local optimum. This phase is called the exploration phase.

After a period of time, the collision between objects doesn't occur as the algorithm tries to get into the equilibrium state. During this time, the algorithm tries to get an optimal solution in the already identified promising region or the local solution space. Like the other population-based metaheuristic algorithms, the Archimedes Optimization algorithm starts its search process with an initial population of objects. These objects in the population

are the candidate solutions for the workflow scheduling problem. Each object is initialized with random densities, volumes, and accelerations. Also, each object is initialized with a random position in the fluid. The densities and volumes of the objects are updated in each iteration of the algorithm. Based on these densities, volumes, and the number of iterations, the accelerations of the objects are updated. Further, the accelerations of the objects are normalized. Based on these normalized accelerations of the objects, the positions of the objects are updated. Following is a detailed explanation of the algorithmic steps and the mathematical formulation for this algorithm.

Object Encoding: An object is a candidate solution to the workflow scheduling problem. It should have an encoding that perfectly includes all the elements that explain the object's behavior, like position, density, volume, acceleration, normalized acceleration, makespan, cost, and fitness. Figure 3 explains the structure of the encoded object we proposed to solve the workflow scheduling problem.

Here, an object comprises eight fields, as shown in Figure 3, of which five are vectors: Position, Density, Volume, Acceleration, and Normalized Acceleration. The index of the vector refers to the task's identity, and the position value refers to the virtual machine to which that particular task is associated. For example, Task 1 is associated with the P1 virtual machine according to this particular solution. The size of these vectors equals the number of tasks in the workflow. To create a population of size N, we must generate N such objects.

Initialization: To initialize the population, each object, except for one, is assigned random values. The position vector of the first object is initialized by seeding it with the Heterogeneous Earliest Finish Time (HEFT) solution, which provides the virtual machine (VM) for task mapping for the formulated problem. For the remaining objects, the position vectors are initialized with random integer values within a specified range, as defined in Equation (21). The lower bound of this range, b_l, corresponds to the smallest virtual machine identity, while the upper bound, b_u, is the largest virtual machine identity. It is assumed that if V virtual machines are created, their identities range from 0 to $V-1$.

$$\rho_n = \vartheta \times (b_u - b_l) + b_l \quad where \quad n = 1, 2, 3, \ldots, N \tag{21}$$

Here, ϑ is a random number generated between 0 and 1. Based on the position vector, the tasks are scheduled on the virtual machines associated with it, and hence, the makespan, cost, and fitness values are computed for each object. Apart from the position vector, the remaining vectors are initialized with random values between 0 and 1 as given in Equations (22)–(24).

$$D_n = \vartheta \quad where \quad n = 1, 2, 3, \ldots, N \tag{22}$$

$$V_n = \vartheta \quad where \quad n = 1, 2, 3, \ldots, N \tag{23}$$

$$A_n = \vartheta \times (b_u - b_l) + b_l \quad where \quad n = 1, 2, 3, \ldots, N \tag{24}$$

Fitness Function: In metaheuristic algorithms, the fitness function is a measure to quantify the quality of the solutions. Depending on the objectives, any problem can have multiple fitness functions. As we have taken makespan and cost minimization as our objectives while finding the solution to the workflow scheduling problem, we have designed the fitness function as given in Equation (25).

$$F_n = \frac{1}{1 + (\alpha) \times M_n + (1-\alpha) \times Cost_n} \quad where \quad n = 1, 2, 3, \ldots, N \tag{25}$$

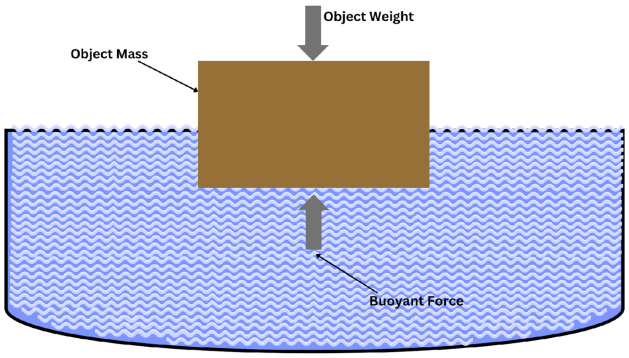

FIGURE 2 | Archimedes principle.

Object									
Position	P1	P2	P3	P4	P5	P6	P7	P8	....	Pn
Density	D1	D2	D3	D4	D5	D6	D7	D8	....	Dn
Volume	V1	V2	V3	V4	V5	V6	V7	V8	....	Vn
Acceleration	A1	A2	A3	A4	A5	A6	A7	A8	....	An
Normalized Acceleration	N1	N2	N3	N4	N5	N6	N7	N8	....	Nn
Makespan										
Cost										
Fitness										

FIGURE 3 | Encoded object.

Here, the value of α decides the significance we want to give to the makespan and cost values. Since we wanted to devise a cost-efficient algorithm, we have taken the alpha value to be 0.4. It prioritizes solutions with lower costs while considering the makespan as well. Hence, this function is well suited for our objective of devising a cost-efficient multi-objective approach. After the initialization of the population and after each iteration, the fitness for each object is computed using the fitness function. The object with the best fitness value is assigned as the O_b, and the position, density, volume, and acceleration associated with that object is assigned as the ρ_b, D_b, V_b and A_b.

Transfer Factor: As discussed earlier, the Archimedes Optimization algorithm goes through two phases to get the best solution, i.e., the exploration and exploitation phase. Based on the transfer factor, the algorithm decides whether to perform the exploration or exploitation phases, i.e., whether to search for a solution in the global solution space or local solution space. The transfer factor depends on the maximum number of iterations and the number of iterations reached so far and can be computed as Equation (26).

$$TF^{m+1} = \exp\left(\frac{(m+1) - M}{M}\right) \tag{26}$$

The transfer factor gradually increases with the number of iterations reached so far. If the transfer factor is below a threshold value, the algorithm enters the exploration phase and explores the possible solutions in the global solution space. As the number of iterations increases and the value of the transfer factor crosses that threshold value, the algorithm tries to stabilize the solution space and find a solution better than the current one in the local solution space or the already identified promising region. Hence, the transfer factor and the threshold value will decide after which iteration the algorithm will shift from the exploration phase to the exploitation phase, i.e., global to local search.

Density Decreasing Factor: The density decreasing factor is another factor that depends on the number of iterations reached and the maximum number of iterations. It is given by the Equation (27).

$$DF^{m+1} = \exp\left(\frac{M - (m+1)}{M}\right) - \left(\frac{1}{M}\right) \tag{27}$$

With the increasing number of iterations reached, the value of the density decreasing factor decreases gradually. This gradual decrease in the value helps the solutions to converge in the already identified promising region. Properly handling this variable positively affects the exploration and exploitation phases.

Update Mechanisms: In each iteration, the transfer factor and the density decreasing factor are computed. Afterward, the density and the volume of the object are updated. Based on the transfer factor and the threshold value, it is decided whether to enter the exploration or exploitation phase. Accordingly, the acceleration value is updated and normalized. The position of the object is updated. Again, the fitness is computed, and the O_b is identified. The following are the update mechanisms.

Best Object Update: Based on the position vector, each object is evaluated as per the fitness function given in Equation (25) and the O_b is updated according to Equation (28).

$$O_b = \begin{cases} O_n & if \quad F_n \geq F_b \\ O_b & Otherwise \end{cases} \tag{28}$$

Density Update: The density of each object is updated as given in Equation (29). The updated density is the previous density with a slight shift towards the density associated with the O_b in the previous iteration.

$$D_n^{m+1} = D_n^m + \vartheta \times (D_b - D_n^m) \tag{29}$$

Volume Update: The volume of each object is updated as given in Equation (30). Also, the updated volume is the previous volume with a slight shift towards the volume associated with the O_b in the previous iteration.

$$V_n^{m+1} = V_n^m + \vartheta \times (V_b - V_n^m) \tag{30}$$

Acceleration Update: Based on the transfer factor, the phase the algorithm will enter will be decided. If the algorithm enters the exploration phase, the acceleration will be updated as in Equation (31). In this phase, a collision between objects happens. Each object O_n collides with any random object O_ϑ, which leads to the change in the acceleration of the O_n.

$$A_n^{m+1} = \frac{D_\vartheta + V_\vartheta \times A_\vartheta}{D_n^{m+1} \times V_n^{m+1}} \tag{31}$$

If the algorithm enters the exploitation phase, the acceleration will be updated according to Equation (32). In this phase, the algorithm tries to get a better solution in an already identified promising region. It updates the acceleration according to the O_b identified so far.

$$A_n^{m+1} = \frac{D_b + V_b \times A_b}{D_n^{m+1} \times V_n^{m+1}} \tag{32}$$

Acceleration Normalization: Acceleration normalization is performed according to Equation (33) to determine the percentage of change in the acceleration as per the minimum and maximum acceleration.

$$NA_n^{m+1} = u \times \frac{A_n^{m+1} - \min(A)}{\max(A) - \min(A)} + l \tag{33}$$

When the algorithm is in the exploration phase, the objects usually have a higher acceleration value. With the increase in the number of iterations reached the acceleration value decreases as the algorithm reaches the exploitation phase. Hence, this technique allows the algorithm to find the best global solution while avoiding getting trapped in local optima. Hence, the algorithm attains the balance between exploration and exploitation.

Position Update: After getting the acceleration value, it's the turn to update the object's position. Depending on the transfer factor and the threshold value, the phase for the current iteration of the algorithm is decided. If the algorithm is in the exploration phase, the position is updated according to the Equation (34). Here, C_1 is a constant value 2.

$$\rho_n^{m+1} = \rho_n^m + C_1 \times \vartheta \times NA_n^{m+1} \times DF^{m+1} \times (\rho_\vartheta - \rho_n^m) \tag{34}$$

ALGORITHM 1 | Modified Archimedes Optimization Algorithm.

Input: TaskList, VmList, Maximum iterations t_{max}, **Population Size** P, **Threshold** θ.
Output: Task to VM mapping with best fitness value

Generate the objects according to the population size.
Initialize the position vector of the first object by seeding the HEFT solution for the problem.
for remaining each $Object_i$ in the population P **do**
    Initialize the Object's position, density, volume, and acceleration according to Equations (21)–(24), respectively.
    Compute the Object's fitness according to Equation (25).
end for
Assign the $Object_{best}$ according to the Equation (28).
while $t < t_{max}$ **do**
    Compute the Transfer Factor TF using the Equation (26).
    Compute the Density Decreasing factor DDF using the Equation (27).
    for each $Object_i$ in the population P **do**
        Update the Object's density using Equation (29).
        Update the Object's volume using Equation (30).
        if $(TF <= \theta)$ **then** **Exploration Phase**
            Update Object's acceleration using Equation (31).
            Normalize Object's acceleration using Equation (33).
            Update Object's position using Equation (34).
        else **Exploitation Phase**
            Update Object's acceleration using Equation (32).
            Normalize Object's acceleration using Equation (33).
            Compute the flag F, S, and T using Equations (36)–(38).
            Update Object's position using Equation (35).
        end if
        Perform the local escaping operation using Algorithm 2 for $Object_i$.
        Compute the Object's fitness according to Equation (25).
    end for
    Update the $Object_{best}$ according to the Equation (28).
    t= t+1
end while

If the algorithm is in the exploitation phase, the position is updated according to the Equation (35).

$$\rho_n^{m+1} = \rho_b^m + F \times C_2 \times \vartheta \times NA_n^{m+1} \times DF^{m+1} \times (T \times \rho_b - \rho_n^m) \tag{35}$$

The variable F denotes a flag that changes the direction of motion and is computed according to Equation (36).

$$F = \begin{cases} +1 & if \ \ S \leq 0.5 \\ -1 & if \ \ S > 0.5 \end{cases} \tag{36}$$

Here, S is the variable computed according to Equation (37).

$$S = 2 \times \vartheta - C_4 \tag{37}$$

where C_4 is a constant value 0.5. The variable T is an increasing variable proportional to the transfer factor and can be computed according to Equation (38).

$$T = C_3 \times TF^{m+1} \tag{38}$$

Here, C_3 is a constant of value 2.

4.1.2 | Local Escaping Operation

The LEO algorithm, introduced as a local search method by Ahmadianfar, Bozorg-Haddad and Chu [30], is designed to enhance the performance of optimization algorithms, such as the Gradient-Based Optimizer (GBO), by facilitating the exploration of new regions, which is particularly useful in addressing complex real-world problems. LEO improves solution quality by updating their positions based on certain criteria, helping to avoid local optima and improving convergence. It generates alternative solutions, denoted as O_{LEO}, that exhibit superior performance by leveraging several key solutions: the best-known solution O_{best}, two randomly generated solutions O_{r1} and O_{r2}, and a newly generated random solution O_{k}. Consequently, the solution O_{LEO} can be expressed using Equations (49) and (50), as mathematically formulated in Algorithm 2:

The operation starts with O_n^m, which represents the current position of the considered object, O_{best} is the best-scored position, R_1 is a random value in the range [0, 1], and f_1 and f_2 are uniformly distributed random values in the range $[-1, 1]$. O_{r1}^m and O_{r2}^m are two randomly selected solutions from the population, and O_n^{m1} and O_n^{m2} are two randomly generated solutions from the current population, as shown in Equation (39):

$$O_n^{m1}, O_n^{m2} = b_l + \text{rand}(Dim) \times (b_u - b_l) \tag{39}$$

where b_l and b_u are the lower and upper bounds, respectively, and Dim represents the dimension of the solution, in our case, the number of tasks. Additionally, n and m denote the coordinates

ALGORITHM 2 | Local Escaping Operation.

Input: Best-known solution O_{best}, **current solution** O_n^m.
Output: Task to VM mapping with best fitness value O_{LEO}.

1: Select two random solutions O_{r1}^m and O_{r2}^m from the current population.
2: Randomly generate two solutions O_n^{m1} and O_n^{m2} from the current population using Equation (39).
3: Randomly generate three variables u_1, u_2 and u_3 as given in Equations (40)–(42).
4: Compute β, α and ρ_1 as given in Equations (45), (44), and (43), and respectively.
5: Determine the solution O_k^m using the Equations (46)–(48).
6: Generate a random number R_1.
7: **if** $R_1 < 0.4$ **then**
8: Update O_m^{LEO} using Equation (49).
9: **else** Update O_m^{LEO} using Equation (50).
10: Update the current solution using Equation (51).

of the solution, where $n = 1, 2, 3, \ldots, N$ and $m = 1, 2, 3, \ldots, Dim$. Furthermore, u_1, u_2, and u_3 are three variables randomly generated as in the Equations (40)–(42):

$$u_1 = L_1 \times 2 \times \text{rand} + (1 - L_1) \tag{40}$$

$$u_2 = L_1 \times \text{rand} + (1 - L_1) \tag{41}$$

$$u_3 = L_1 \times \text{rand} + (1 - L_1) \tag{42}$$

where L_1 is a binary parameter (i.e., $L_1 = 1$ if $\mu_1 < 0.5$, otherwise $L_1 = 0$), and μ_1 is a random value in the range $[0, 1]$. Moreover, ρ_1 is introduced to balance the exploration and exploitation phases during the search process, and it can be expressed as in Equation (43):

$$\rho_1 = 2 \times \text{rand} \times \alpha - \alpha \tag{43}$$

where α can be computed using Equation (44) and β can be computed as in Equation (45).

$$\alpha = \left| \beta \times \sin\left(\frac{3\pi}{2} + \sin\left(\beta \times \frac{3\pi}{2} \right) \right) \right| \tag{44}$$

$$\beta = \beta_{\min} + (\beta_{\max} - \beta_{\min}) \times \left(1 - \left(\frac{t}{t_{\max}} \right)^3 \right)^2 \tag{45}$$

Here, $\beta_{\min}$ and $\beta_{\max}$ are set to 0.2 and 1.2, respectively, t is the current iteration, and $t_{\max}$ is the maximum number of iterations. The parameter ρ_1 varies according to the sine function α, ensuring a balance between exploration and exploitation. Further, to determine the solution O_k^m, the following Equations (46) and (48) are used.

$$O_k^m = \begin{cases} O_{\text{rand}} & \text{if } \mu_2 < 0.5 \\ O_p^m & \text{otherwise} \end{cases} \tag{46}$$

Here, O_{rand} is a new solution calculated using Equation (47), O_p^m is a random solution selected from the population ($p \in [1, 2, \ldots, N]$), and μ_2 is a random number in the range $[0, 1]$.

$$O_{\text{rand}} = b_l + \text{rand}(0, 1) \times (b_u - b_l) \tag{47}$$

Equation (46) can be reformulated as Equation (48):

$$O_k^m = L_2 \times O_p^m + (1 - L_2) \times O_{\text{rand}} \tag{48}$$

Here L_2 is a binary parameter that takes the value 0 or 1. If the parameter μ_2 is less than 0.5, L_2 is set to 1; otherwise, it is set to 0. Further, a random value is generated between 0 and 1.If $R_1 < 0.4$, then the solution O_m^{LEO} is updated according to Equation (49), otherwise it is updated according to Equation (50).

$$O_m^{LEO} = O_n^m + f_1 \times \left(u_1 \times O_{\text{best}} - u_2 \times O_k^m \right) + f_2 \times \rho_1 \left(u_3 \times \left(O_n^{m2} - O_n^{m1} \right) \right) + \frac{u_2 \times \left(O_{r1}^m - O_{r2}^m \right)}{2} \tag{49}$$

$$O_m^{LEO} = O_{\text{best}} + f_1 \times \left(u_1 \times O_{\text{best}} - u_2 \times O_k^m \right) + f_2 \times \rho_1 \times \left(u_3 \times \left(O_n^{m2} - O_n^{m1} \right) \right) + \frac{u_2 \times \left(O_{r1}^m - O_{r2}^m \right)}{2} \tag{50}$$

$$O_n^{m+1} = O_m^{LEO} \tag{51}$$

Finally, the current solution is updated according to the Equation (51) using the local escaping operation.

4.2 | The MLEAO Approach

Like many other optimization algorithms, the Archimedes Optimization Algorithm (AOA) faces the challenge of getting stuck in local optima. Being trapped in a local region can prevent the optimizer from reaching the global optimal solution. This issue is particularly common in certain optimization scenarios, especially in complex and high-dimensional problems [31]. Additionally, the generation of new solutions is based on those from the previous iteration. This approach may slow down the convergence rate of the algorithm and prevent solutions from effectively covering the entire search space, leading to premature convergence. Similar to most optimization algorithms, the Improved Arithmetic Optimization Algorithm (MLEAO) begins with an initialization process by generating an initial population consisting of N objects. Each search object has a dimensionality Dim (number of tasks) within the search space, bounded by the upper and lower limits (b_u and b_l representing the maximum and minimum identity number of virtual machines).

The complete process is discussed in Algorithms 1 and 2. It starts with the list of VMs and tasks. The objects in the population are generated according to the given population size. Thereafter, the whole population is initialized as given in Algorithm 1. The position vector of the first solution is initiated by seeding the HEFT solution for the problem. For remaining objects, the position vector is initialized as discussed in the Equation (21). The density, volume, and acceleration vectors of all the objects are initialized according to the Equations (22)–(24). Based on the current position vector, the makespan, and the cost are computed; hence, each object's fitness is evaluated according to Equation (25). Then, the object with the best fitness value is assigned as the $Object_{\text{best}}$ according to Equation (28). Afterward, the algorithm iterations start. For each iteration, the transfer and density decreasing factor is computed as in Equations (26) and

(27). For each object in the population, the density and the volume are updated according to the Equations (29) and (30).

Based on the transfer factor, whether the algorithm will enter the exploration or exploitation phase is decided. In the exploration phase, acceleration is updated as in Equation (31); otherwise, Equation (32). Afterward, the acceleration is normalized as given in Equation (33). In the exploration phase, the position is updated according to the Equation (34). Otherwise, the flag F, S and T are computed as in Equations (36)–(38), and further, the position is updated as given in Equation (35). Afterward, the current solution is passed on to the Algorithm 2 for the local escaping operation. Algorithm 2 randomly selects two solutions from the current population. Afterward, it generates two random solutions from the current population using Equation (39). Further, It randomly generates three variables u_1, u_2 and u_3 according to the Equations (40)–(42). Then, β, α and ρ_1 are computed as given in Equations (45), (44), and (43) respectively. Further, the solution O_k^m is defined using the Equations (46)–(48). Afterward, a random number R_1 is generated between 0 and 1. If $R_1 < 0.4$, O_m^{LEO} is updated using Equation (49) otherwise using Equation (50). Then the current solution is updated using Equation (51). Then, Algorithm 2 returns the current solution obtained after performing the local escaping operation to the Algorithm 1. Now, the fitness of the returned solution is calculated. At the end of each iteration, the $Object_{\text{best}}$ is updated as per the Equation (28). When the maximum number of iterations is reached, the position vector of the $Object_{best}$ is taken as the final allocation of tasks on the virtual machine, and hence, the algorithm concludes.

5 | Experimental Environment & Performance Evaluation

5.1 | Experimental Setup

This section discusses the extensive experiments performed over the five well-known scientific workflows. The experiments were performed on the well-known cloud simulation tool, named Workflowsim, using Eclipse IDE on a desktop with the system type 64-bit, Windows 10 Pro operating system, x64-based processor Intel(R) Core(TM) i7-7700 CPU 3.60 GHz with installed RAM 32.00 GB (31.9 GB usable).

We have performed extensive experiments on the five well-known workflows with nearly 30, 50, 100, and 1,000 tasks each for this purpose: Montage, SIPHT, CyberShake, Epigenomics, and Inspiral [32]. We scheduled these tasks on 20 virtual machines with specifications similar to Amazon EC2 C4 instances as described in Table 3.

TABLE 3 | VM instances.

VM Instance	Number of vCPUs	Memory (GiB)	Price ($)
c4.large	2	3.75	0.116
c4.xlarge	4	7.5	0.232
c4.2xlarge	8	15	0.464
c4.4xlarge	16	30	0.928
c4.8xlarge	36	60	1.856

5.2 | Results

We have compared our proposed work, Cost-efficient Archimedes Optimization (MLEAO) approach with well-known workflow scheduling algorithms, i.e., EARES [33], Genetic Algorithm (GA) [34], Particle Swarm Optimization (PSO) [35] and Modified Fuzzy Genetic Algorithm (MFGA) [36]. Here, we have taken the basic GA and replaced the fitness function with the fitness function used in our approach to make it multi-objective and have a fair competition. Similarly, we have done for PSO. MFGA improved GA by introducing a fuzzy logic scheme to update the crossover and mutation parameters after each iteration. We have compared our approach with these well-known algorithms.

5.2.1 | Parameter Settings

The various parameters taken for the experiments are discussed in this section. The average network bandwidth was 20 Mbps, and its cost per bandwidth was 0.1$. We performed 50 simulations of all the algorithms on all the workflows discussed earlier. Each simulation had 100 iterations with a population size of 50. To decide the number of iterations to be performed while simulation, we performed rigorous simulations on various numbers of iterations for all the algorithms. Figure 4 depicts the convergence graph of state-of-the-art approaches for various workflows when compared to our proposed approach. All the compared approaches are converging around 100 iterations, but our approach is converging around 90 iterations. To maintain an upper bound, the number of iterations for which the final simulations were performed is taken as 100. For GA, the crossover and mutation rates were set to be 70% and 30%. For PSO, the parameters c_1 and c_2 were equal to 2, and the value of ω was set to 0.5. For MFGA, the starting crossover and mutation rates were taken to be 70% and 30%, respectively, which were further updated by the algorithm. For our approach, we have taken the C_1, C_2, C_3 and C_4 as 2, 6, 2, and 0.5, respectively. To determine the values of these parameters, we have performed a sensitivity analysis presented in Table 4. It presents the fitness values obtained for various workflows by considering various feasible values for these parameters as provided by Hashim et al. [29]. The best fitness values for all the workflows were obtained in scenario 23; that's why it is considered the final parameter value in our approach. As we wanted our proposed approach to be cost-effective, the value of α was taken as 0.4 thoroughly. The threshold θ for the exploration and exploitation phase decision was taken as 0.5.

5.2.2 | Evaluation Metrics

We have compared our approach to the state-of-the-art algorithms based on evaluation metrics such as makespan, cost, energy consumption, and resource utilization. Rather than just using these metrics for performance comparison of our approach to the other algorithms, we have also compared the metaheuristic approaches based on Pareto-optimality metrics, such as hypervolume, s-metric, and dominance, as our objectives were makespan and cost efficiency. In the end, we also discussed the overall complexity analysis of the various approaches.

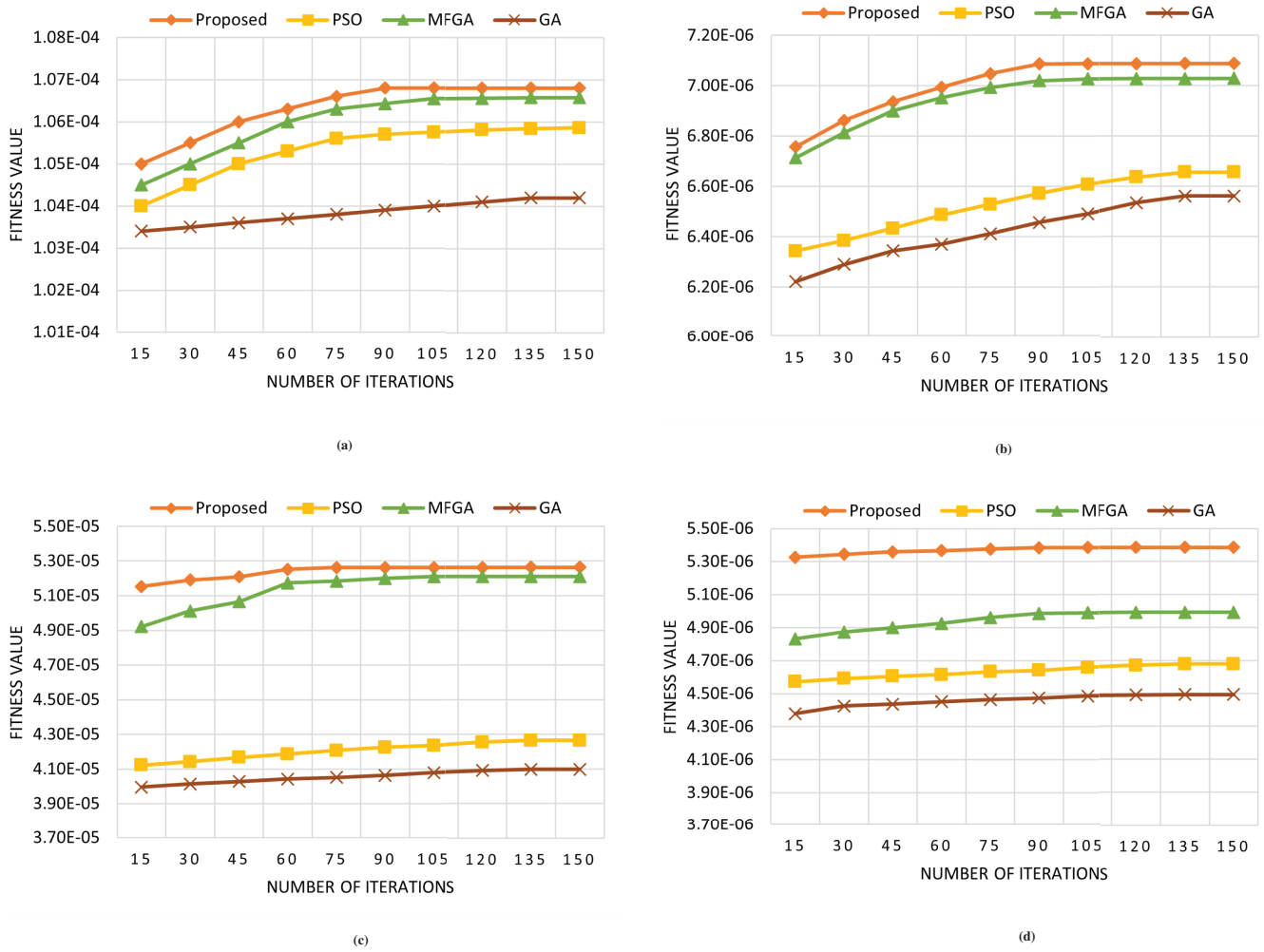

FIGURE 4 | Convergence analysis of state of the art for various workflows. (a) Montage. (b) SIPHT. (c) CyberShake. (d) Inspiral.

Makespan, Cost, Energy Consumption and Resource Utilization: To compare the approaches according to their makespan, cost, energy consumption, and resource utilization values, we have computed these metrics for the final schedule of each simulation according to Equations (9), (15), (18), and (19), respectively. Further, we have taken the average values after performing all the simulations. Figure 5a–d presents the average makespan, cost, energy consumption, and resource utilization, respectively, for the Montage workflow with 25, 50, and 100 tasks.Figure 6a–d presents the average makespan, cost, energy consumption, and resource utilization, respectively, for the SIPHT workflow with 30, 60, and 100 tasks. Figure 7a–d presents the average makespan, cost, energy consumption, and resource utilization, respectively, for the CyberShake workflow with 30, 50, and 100 tasks. Figure 8a–d presents the average makespan, cost, energy consumption, and resource utilization, respectively, for the Inspiral LIGO workflow with 30, 50, and 100 tasks. Figure 9a–d presents the average makespan, cost, energy consumption, and resource utilization, respectively, for the Epigenomics workflow with 24, 46, and 100 tasks. Figure 10a–d presents the average makespan, cost, energy consumption, and resource utilization, respectively, for the Montage and CyberShake workflows with 1,000 tasks. Figure 11a–d presents the average makespan, cost, energy consumption, and resource utilization, respectively, for the SIPHT and Inspiral LIGO workflows with 1,000 tasks. Figure 12a–d presents the average makespan, cost, energy consumption, and resource utilization, respectively, for the Epigenomics workflow with 997 tasks.

Hypervolume: The volume between a set of solutions and a reference point is used to characterize the diversity and convergence of the solutions of any approach and is known as hypervolume. Concretely, a higher hypervolume value means higher quality and distribution of the obtained solutions [37]. Table 5 compares the hypervolume values of the state-of-the-art algorithms to our approach on a given reference point.

$$D(A, B) = \frac{|b \in B, \exists a \in A, a \geq b|}{|B|} \tag{52}$$

Dominance: The dominance relationship between two solution sets is denoted as in Equation (52), in which the ratio of solutions in B dominated by solutions in A is represented by D(A, B). Table 6 compares our approach's dominance relationship with other approaches. For example, D(MLEAO, GA) represents the dominance of MLEAO over GA.

TABLE 4 | Sensitivity analysis of different parameters for various workflows.

Scenario	C1	C2	C3	C4	Montage	Sipht	CyberShake	Inspiral	Epigenomics
1	1	2	1	0.5	0.000105217	7.0185E−06	5.22919E−05	5.35008E−06	3.21556E−07
2	1	2	1	1	0.000104748	7.03625E−06	5.23939E−05	5.35031E−06	3.22103E−07
3	1	2	2	0.5	0.000104881	6.94986E−06	5.2258E−05	5.35062E−06	3.21581E−07
4	1	2	2	1	0.000105073	7.04032E−06	5.20995E−05	5.36382E−06	3.22038E−07
5	1	4	1	0.5	0.000105199	7.03073E−06	5.22569E−05	5.34933E−06	3.21604E−07
6	1	4	1	1	0.000104737	6.98677E−06	5.25189E−05	5.33561E−06	3.21178E−07
7	1	4	2	0.5	0.000105002	7.03647E−06	5.23454E−05	5.35584E−06	3.22526E−07
8	1	4	2	1	0.000105179	7.03007E−06	5.25138E−05	5.35573E−06	3.21768E−07
9	1	6	1	0.5	0.000105244	7.00405E−06	5.25329E−05	5.34938E−06	3.22365E−07
10	1	6	1	1	0.000105146	7.04167E−06	5.20887E−05	5.356E−06	3.21318E−07
11	1	6	2	0.5	0.000105182	7.04017E−06	5.2417E−05	5.35793E−06	3.22517E−07
12	1	6	2	1	0.000105358	6.99816E−06	5.24434E−05	5.36234E−06	3.18566E−07
13	2	2	1	0.5	0.000105032	7.02749E−06	5.23241E−05	5.36921E−06	3.21577E−07
14	2	2	1	1	0.000105285	6.96698E−06	5.23224E−05	5.37479E−06	3.22029E−07
15	2	2	2	0.5	0.00010542	7.02052E−06	5.2263E−05	5.37045E−06	3.22026E−07
16	2	2	2	1	0.000105435	7.08169E−06	5.2212E−05	5.3579E−06	3.20806E−07
17	2	4	1	0.5	0.000105415	7.04787E−06	5.23327E−05	5.3469E−06	3.22211E−07
18	2	4	1	1	0.000105393	7.02593E−06	5.23798E−05	5.36903E−06	3.21678E−07
19	2	4	2	0.5	0.000105468	7.04272E−06	5.2449E−05	5.36477E−06	3.22036E−07
20	2	4	2	1	0.000105452	7.01401E−06	5.25193E−05	5.35295E−06	3.21996E−07
21	2	6	1	0.5	0.000105583	7.01939E−06	5.25455E−05	5.36293E−06	3.22148E−07
22	2	6	1	1	0.000105674	6.98326E−06	5.25407E−05	5.37892E−06	3.21481E−07
23	**2**	**6**	**2**	**0.5**	**0.000105842**	**7.08873E − 06**	**5.26369E − 05**	**5.38544E − 06**	**3.22719E − 07**
24	2	6	2	1	0.000105791	7.07131E−06	5.22584E−05	5.37661E−06	3.21497E−07

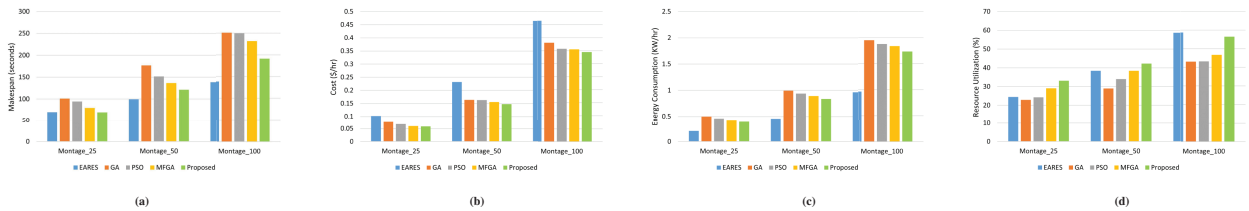

FIGURE 5 | Montage. (a) Makespan. (b) Cost. (c) Energy consumption. (d) Resource utilization.

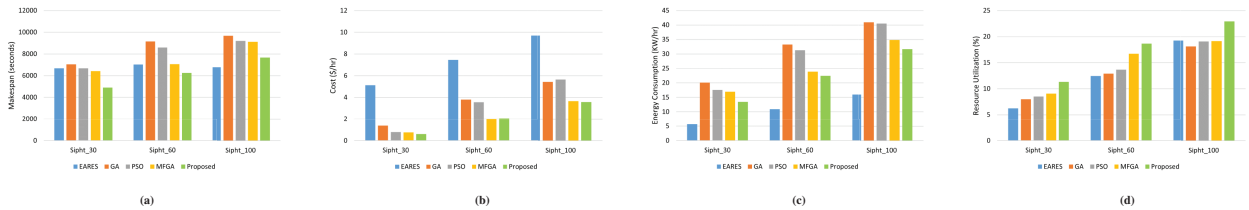

FIGURE 6 | SIPHT. (a) Makespan. (b) Cost. (c) Energy consumption. (d) Resource utilization.

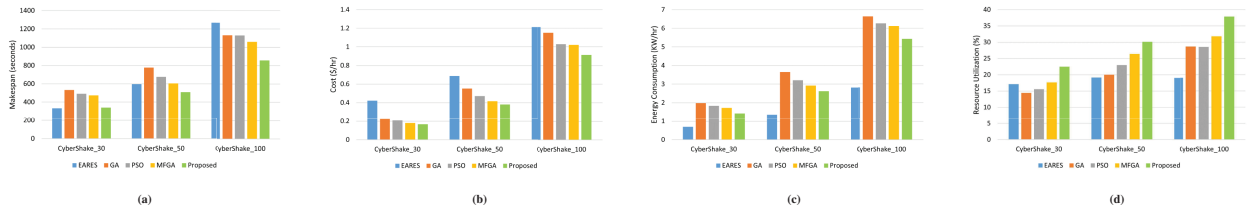

FIGURE 7 | CyberShake. (a) Makespan. (b) Cost. (c) Energy consumption. (d) Resource utilization.

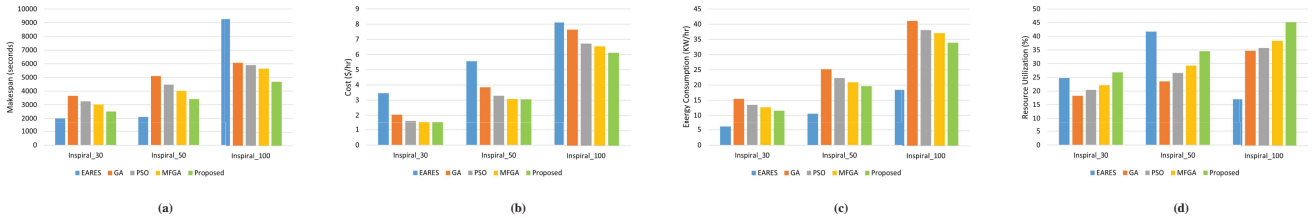

FIGURE 8 | Inspiral LIGO. (a) Makespan. (b) Cost. (c) Energy consumption. (d) Resource utilization.

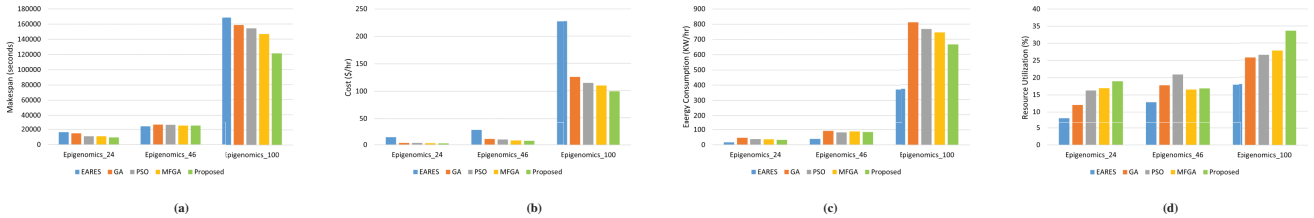

FIGURE 9 | Epigenomics. (a) Makespan (b) Cost. (c) Energy consumption. (d) Resource utilization.

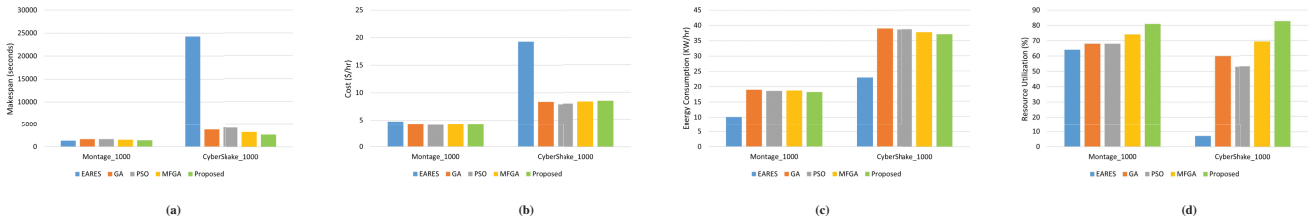

FIGURE 10 | Montage_1000 and CyberShake_1000. (a) Makespan. (b) Cost. (c) Energy consumption. (d) Resource utilization.

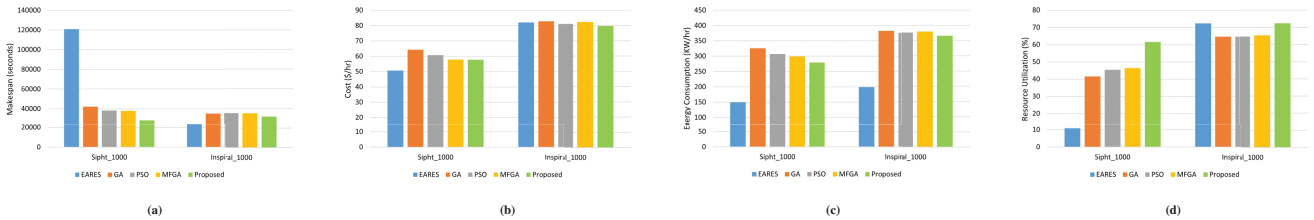

FIGURE 11 | SIPHT_1000 and Inspiral_1000. (a) Makespan. (b) Cost. (c) Energy consumption. (d) Resource utilization.

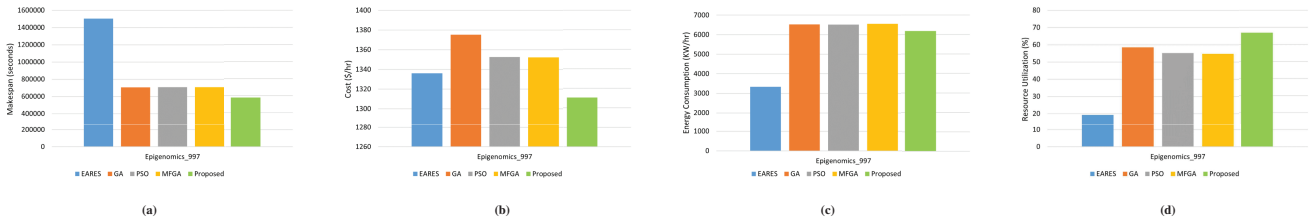

FIGURE 12 | Epigenomics_997. (a) Makespan. (b) Cost. (c) Energy consumption. (d) Resource utilization.

TABLE 5 | Comparison for hypervolume.

Reference Point	Workflow	GA	PSO	MFGA	MLEAO
(500, 500)	Montage_25	1.32E+05	1.36E+05	1.41E+05	**1.43E+05**
(1,000, 1,000)	Montage_50	4.06E+05	4.40E+05	4.45E+05	**4.84E+05**
(2,000, 2,000)	Montage_100	1.47E+06	**1.67E+06**	1.38E+06	1.52E+06
(18,000, 18,000)	Montage_1000	6.04E+07	**6.92E+07**	5.10E+07	6.07E+07
(15,000, 15,000)	Sipht_30	1.19E+08	1.32E+08	1.28E+08	**1.35E+08**
(25,000, 25,000)	Sipht_60	3.13E+08	3.02E+08	**3.60E+08**	3.57E+08
(250,000, 250,000)	Sipht_100	5.74E+10	5.76E+10	5.75E+10	**5.86E+10**
(2,500,000, 2,500,000)	Sipht_1000	5.67E+12	5.69E+12	5.66E+12	**5.71E+12**
(1,000, 1,500)	CyberShake_30	5.29E+05	5.89E+05	6.13E+05	**6.47E+05**
(2,000, 3,000)	CyberShake_50	2.12E+06	2.59E+06	2.41E+06	**2.84E+06**
(10,000, 15,000)	CyberShake_100	1.05E+08	1.09E+08	1.09E+08	**1.11E+08**
(50,000, 50,000)	CyberShake_1000	9.97E+08	**1.04E+09**	9.56E+08	9.60E+08
(50,000, 50,000)	Inspiral_30	2.14E+09	2.16E+09	2.14E+09	**2.20E+09**
(50,000, 50,000)	Inspiral_50	1.85E+09	1.88E+09	1.83E+09	**1.89E+09**
(50,000, 50,000)	Inspiral_100	1.29E+09	1.27E+09	1.25E+09	**1.38E+09**
(50,000, 500,000)	Inspiral_1000	3.52E+09	3.50E+09	3.61E+09	**4.50E+09**
(50,000, 500,000)	Epigenomics_24	1.82E+10	1.97E+10	2.06E+10	**2.07E+10**
(50,000, 500,000)	Epigenomics_46	1.38E+10	**1.46E+10**	1.32E+10	1.41E+10
(500,000, 1,500,000)	Epigenomics_100	4.27E+11	4.58E+11	4.18E+11	**4.74E+11**
(5,000,000, 5,500,000)	Epigenomics_997	**4.56E+12**	3.61E+12	3.06E+12	4.04E+12

S-Metric: It is the metric applied to quantify the uniformity of a set of achieved solutions. Hence, the lower the value of the s-metric for several simulations, the better the solution quality. Table 7 compares our approach's s-metric values for various workflows to the state-of-the-art approaches. The definition of s-metric is given by Equation (53):

$$S = \sqrt{\frac{1}{N_p} \times \sum_{i=1}^{N_p} (d_i - \overline{d})^2} \tag{53}$$

where N_p is the number of solutions and $\overline{d}$ can be computed as in Equation (54):

$$\overline{d} = \frac{1}{N_p} \sum_{i=1}^{N_p} \times d_i \tag{54}$$

5.2.3 | Performance Observations

As illustrated in Figure 5a–d, our method outperforms all metaheuristics in most cases. However, while our approach generally maintains steady performance, it does not surpass the heuristic-based EARES approach in numerous instances. EARES, though showing inconsistent performance, excelled in certain workflows compared to metaheuristic approaches but performed worse in others. Overall, our proposed method achieved a 34% improvement in makespan, 36% in cost, and 26% in energy consumption over GA. Similarly, compared to PSO, our approach reduced makespan by 26%, cost by 22%, and energy by 17%. Against MFGA, we observed a 17% improvement in makespan, 8% in cost, and 11% in energy consumption. In our investigations, we found that for our proposed work, for about 95% of cases, we got better resource utilization than the state-of-the-art approaches. After analyzing Table 5, we found that we got a higher hypervolume for about 70% of cases than the compared algorithms, which shows that the set of solutions generated by our algorithms is diverse. Hence, we are not stuck in local optima. After analyzing Table 7, we found that our approach got a lower s-metric value for about 95% of cases than the compared algorithms, which shows that the set of solutions generated by our algorithms is uniform, which means any two nearest points in our solution set don't lie too far from each other. From Table 6, we observed that our method dominated GA by 94.6% on average, while GA could dominate our solutions by 4.29%. Our approach dominated the PSO algorithm by 82.11% on average, while the PSO algorithm could dominate our solutions by 10.20%. Our approach dominated the MFGA algorithm by 90.67% on average, while the MFGA algorithm could dominate our solutions by 52.4%. The Archimedes optimization algorithm achieves a robust balance between the exploration and exploitation phases, refining the solution space by emphasizing promising areas. This balance contributes to superior solution quality relative to other advanced approaches. Additionally, the fitness function prioritizes cost optimization, while initializing with the HEFT solution as the position vector for one object steers the population towards solutions with improved makespan. Together, these strategies yield better outcomes in both makespan and cost. The inclusion of a local escaping operation further enhances solution quality by dynamically regulating exploration and exploitation, helping to avoid local optima.

TABLE 6 | Comparison for dominance (in %).

Workflows	(MLEAO, GA)	(GA, MLEAO)	(MLEAO, PSO)	(PSO, MLEAO)	(MLEAO, MFGA)	(MFGA, MLEAO)
Montage_25	100.00	4.00	94.00	8.00	82.00	78.00
Montage_50	100.00	2.00	100.00	6.00	98.00	74.00
Montage_100	100.00	0.00	100.00	16.00	100.00	20.00
Montage_1000	100.00	4.00	88.00	0.00	100.00	0.00
Sipht_30	94.00	2.00	94.00	18.00	94.00	76.00
Sipht_60	96.00	2.00	72.00	2.00	94.00	70.00
Sipht_100	82.00	4.00	84.00	6.00	94.00	78.00
Sipht_1000	98.04	3.92	98.04	1.96	35.29	1.96
CyberShake_30	98.00	2.00	90.00	6.00	98.00	66.00
CyberShake_50	100.00	1.96	90.20	1.96	96.08	62.75
CyberShake_100	100.00	2.00	100.00	8.00	100.00	34.00
CyberShake_1000	100.00	2.00	44.00	0.00	60.00	4.00
Inspiral_30	92.00	0.00	26.00	2.00	92.00	82.00
Inspiral_50	100.00	0.00	80.00	10.00	100.00	60.00
Inspiral_100	84.00	2.00	86.00	2.00	92.00	56.00
Inspiral_1000	100.00	10.00	100.00	4.00	100.00	0.00
Epigenomics_24	94.00	2.00	80.00	18.00	96.00	76.00
Epigenomics_46	78.00	2.00	70.00	14.00	100.00	66.00
Epigenomics_100	100.00	2.00	70.00	10.00	100.00	36.00
Epigenomics_997	76.00	38.00	76.00	70.00	82.00	100.00

TABLE 7 | Comparison for s-metric.

Workflow	GA	PSO	MFGA	MLEAO
Montage_25	1.73E+01	1.76E+01	1.62E+01	**1.42E+01**
Montage_50	4.01E+01	3.47E+01	3.12E+01	**2.53E+01**
Montage_100	4.97E+01	4.62E+01	4.13E+01	**3.30E+01**
Montage_1000	1.67E+02	1.84E+02	1.68E+02	**1.41E+02**
Sipht_30	1.44E+03	1.29E+03	1.24E+03	**1.02E+03**
Sipht_60	2.14E+03	2.06E+03	1.90E+03	**1.58E+03**
Sipht_100	2.73E+03	2.83E+03	2.63E+03	**2.17E+03**
Sipht_1000	9.10E+03	8.47E+03	7.87E+03	**6.48E+03**
CyberShake_30	1.07E+02	9.68E+01	9.06E+01	**7.93E+01**
CyberShake_50	1.39E+02	1.29E+02	1.21E+02	**1.02E+02**
CyberShake_100	3.28E+02	2.84E+02	2.62E+02	**2.09E+02**
CyberShake_1000	4.23E+02	4.82E+02	4.34E+02	**3.54E+02**
Inspiral_30	6.25E+02	5.69E+02	5.43E+02	**4.91E+02**
Inspiral_50	1.07E+03	1.04E+03	9.78E+02	**8.10E+02**
Inspiral_100	1.01E+03	1.06E+03	9.91E+02	**8.88E+02**
Inspiral_1000	4.67E+03	5.01E+03	4.91E+03	**4.07E+03**
Epigenomics_24	2.39E+03	2.26E+03	2.04E+03	**1.72E+03**
Epigenomics_46	3.23E+03	3.73E+03	3.39E+03	**2.94E+03**
Epigenomics_100	2.09E+04	2.23E+04	2.15E+04	**1.76E+04**
Epigenomics_997	**1.05E+05**	1.48E+05	1.30E+05	1.16E+05

5.3 | Complexity Analysis

The time complexity for all the metaheuristic algorithms we discussed is similar, i.e., $O(n^3)$. However, the computational complexity of GA is more prominent than our method as it includes additional methods for performing the mutation and crossover. MFGA is an advanced version of GA with all the computational complexities of GA and the additional complexity of implementing fuzzy logic in each iteration. Our approach implements some arithmetic operations to get the new position for the object. Hence, similar to PSO, the computational complexity in our case is lower than that of the other algorithms.

6 | Conclusions & Future Research Directions

The enormous increase in usage of cloud computing has led to extensive research in the workflow scheduling area. Numerous algorithms have been proposed to schedule workflows for the cloud computing paradigm. Nevertheless, most scheduling algorithms deal with scheduling a workflow around a single specific objective, usually makespan. Regarding user requirements, cost is also a significant aspect of the cloud computing paradigm. The likelihood of customers utilizing the services offered by cloud service providers increases with decreasing costs. This article addresses the limitations of heuristic, metaheuristic, and hybrid workflow scheduling approaches, which frequently suffer from local optima entrapment, particularly under heavy cloud traffic, and highlights the need for less computationally complex methods. To meet this need, it proposes a new multi-objective Modified Local Escaping Archimedes Optimization (MLEAO) algorithm. This approach initializes the Archimedes Optimization algorithm's population using the HEFT algorithm to guide solutions toward improved makespan while achieving cost-efficient workflow scheduling and mitigating local optima entrapment through a local escaping operation. The experiments were conducted using the well-known scientific workflows Montage, SIPHT, CyberShake, Epigenomics, and Inspiral. Numerous metrics, including the average makespan, cost, resource usage, and energy consumption, were compared with the state-of-the-art multi-objective approaches. A significant improvement was observed in these metrics. Moreover, the effectiveness of the proposed approach is also verified by performance metrics such as hypervolume, s-metric, and dominance relationships between the proposed and state-of-the-art approaches.

In the future, other scheduling objectives, such as load balancing, energy efficiency, and fault tolerance, can be considered when making scheduling decisions. Moreover, constraints like deadlines and budget can be considered for future research work.

Data Availability Statement

The authors have nothing to report.

References

1. Y. Wang and P. Lu, "Dataflow Detection and Applications to Workflow Scheduling," *Concurrency and Computation: Practice and Experience* 23, no. 11 (2011): 1261–1283.

2. L. Zhao, Y. Ren, and K. Sakurai, "Reliable Workflow Scheduling With Less Resource Redundancy," *Parallel Computing* 39, no. 10 (2013): 567–585.

3. M. Sanaj and P. J. Prathap, "An Enhanced Round Robin (ERR) Algorithm for Effective and Efficient Task Scheduling in Cloud Environment," in *2020 Advanced Computing and Communication Technologies for High Performance Applications (ACCTHPA)* (Cochin, India: IEEE, 2020), 107–110.

4. S. Tuli, R. Sandhu, and R. Buyya, "Shared Data-Aware Dynamic Resource Provisioning and Task Scheduling for Data Intensive Applications on Hybrid Clouds Using Aneka," *Future Generation Computer Systems* 106 (2020): 595–606.

5. Y. H. Jia, W. N. Chen, H. Yuan, et al., "An Intelligent Cloud Workflow Scheduling System With Time Estimation and Adaptive Ant Colony Optimization," *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51, no. 1 (2018): 634–649.

6. X. Chen, L. Cheng, C. Liu, et al., "A WOA-Based Optimization Approach for Task Scheduling in Cloud Computing Systems," *IEEE Systems Journal* 14, no. 3 (2020): 3117–3128.

7. N. Anwar and H. Deng, "A Hybrid Metaheuristic for Multi-Objective Scientific Workflow Scheduling in a Cloud Environment," *Applied Sciences* 8, no. 4 (2018): 538.

8. A. M. Manasrah and A. H. Ba, "Workflow Scheduling Using Hybrid GA-PSO Algorithm in Cloud Computing," *Wireless Communications and Mobile Computing* 2018, no. 1 (2018): 1934784.

9. X. Ma, H. Xu, H. Gao, and M. Bian, "Real-Time Multiple-Workflow Scheduling in Cloud Environments," *IEEE Transactions on Network and Service Management* 18, no. 4 (2021): 4002–4018.

10. R. Medara and R. S. Singh, "Energy-Aware Workflow Task Scheduling in Clouds With Virtual Machine Consolidation Using Discrete Water Wave Optimization," *Simulation Modelling Practice and Theory* 110 (2021): 102323.

11. R. Medara, R. S. Singh, and M. Sompalli, "Energy and Cost Aware Workflow Scheduling in Clouds With Deadline Constraint," *Concurrency and Computation: Practice and Experience* 34, no. 13 (2022): e6922.

12. A. Taghinezhad-Niar, S. Pashazadeh, and J. Taheri, "Energy-Efficient Workflow Scheduling With Budget-Deadline Constraints for Cloud," *Computing* 104, no. 3 (2022): 601–625.

13. T. P. Pham and T. Fahringer, "Evolutionary Multi-Objective Workflow Scheduling for Volatile Resources in the Cloud. *IEEE Transactions on Cloud*," *Computing* 10 (2020): 1780–1791.

14. R. Medara and R. S. Singh, "Energy Efficient and Reliability Aware Workflow Task Scheduling in Cloud Environment," *Wireless Personal Communications* 119, no. 2 (2021): 1301–1320.

15. S. Singh and R. Kumar, "Energy Efficient Optimization With Threshold Based Workflow Scheduling and Virtual Machine Consolidation in Cloud Environment," *Wireless Personal Communications* 128 (2022): 1–22.

16. S. Sefati, M. Mousavinasab, and F. R. Zareh, "Load Balancing in Cloud Computing Environment Using the Grey Wolf Optimization Algorithm Based on the Reliability: Performance Evaluation," *Journal of Supercomputing* 78, no. 1 (2022): 18–42.

17. M. Sharma and R. Garg, "HIGA: Harmony-Inspired Genetic Algorithm for Rack-Aware Energy-Efficient Task Scheduling in Cloud Data Centers," *Engineering Science and Technology, an International Journal* 23, no. 1 (2020): 211–224.

18. S. E. Shukri, R. Al-Sayyed, A. Hudaib, and S. Mirjalili, "Enhanced Multi-Verse Optimizer for Task Scheduling in Cloud Computing Environments," *Expert Systems with Applications* 168 (2021): 114230.

19. Z. Sun, B. Zhang, C. Gu, R. Xie, B. Qian, and H. Huang, "ET2FA: A Hybrid Heuristic Algorithm for Deadline-Constrained Workflow Scheduling in Cloud," *IEEE Transactions on Services Computing* 16 (2022): 1807–1821.

20. I. Attiya, M. Abd Elaziz, L. Abualigah, T. N. Nguyen, and A. A. Abd El-Latif, "An Improved Hybrid Swarm Intelligence for Scheduling Iot Application Tasks in the Cloud," *IEEE Transactions on Industrial Informatics* 18 (2022): 6264–6272.

21. P. Pirozmand, A. Javadpour, H. Nazarian, P. Pinto, S. Mirkamali, and F. Ja'fari, "GSAGA: A Hybrid Algorithm for Task Scheduling in Cloud Infrastructure," *Journal of Supercomputing* 78, no. 15 (2022): 17423–17449.

22. P. Shobeiri, M. Akbarian Rastaghi, S. Abrishami, and B. Shobiri, "PCP–ACO: A Hybrid Deadline-Constrained Workflow Scheduling Algorithm for Cloud Environment," *Journal of Supercomputing* 80, no. 6 (2024): 7750–7780.

23. R. Ghafari and N. Mansouri, "A Novel Energy-Based Task Scheduling in Fog Computing Environment: An Improved Artificial Rabbits Optimization Approach," *Cluster Computing* 27 (2024): 1–46.

24. B. M. H. Zade, M. M. Javidi, and N. Mansouri, "An Improved Caledonian Crow Learning Algorithm Based on Ring Topology for Security-Aware Workflow Scheduling in Cloud Computing," *Peer-to-Peer Networking and Applications* 16, no. 6 (2023): 2929–2984.

25. A. A. Mary, "Scientific Workflow Scheduling Using Adaptive Dingo Optimization in Multi-Cloud Environment," *International Journal of Information Technology* 16 (2024): 1–8.

26. X. Tang, W. Cao, H. Tang, et al., "Cost-Efficient Workflow Scheduling Algorithm for Applications With Deadline Constraint on Heterogeneous Clouds," *IEEE Transactions on Parallel and Distributed Systems* 33, no. 9 (2021): 2079–2092.

27. S. Nabi and M. Ahmed, "PSO-RDAL: Particle Swarm Optimization-Based Resource-and Deadline-Aware Dynamic Load Balancer for Deadline-Constrained Cloud Tasks," *Journal of Supercomputing* 78 (2022): 1–31.

28. M. Adhikari, T. Amgoth, and S. N. Srirama, "A Survey on Scheduling Strategies for Workflows in Cloud Environment and Emerging Trends," *ACM Computing Surveys* 52, no. 4 (2019): 1–36.

29. F. A. Hashim, K. Hussain, E. H. Houssein, M. S. Mabrouk, and W. Al-Atabany, "Archimedes Optimization Algorithm: A New Metaheuristic Algorithm for Solving Optimization Problems," *Applied Intelligence* 51 (2021): 1531–1551.

30. I. Ahmadianfar, O. Bozorg-Haddad, and X. Chu, "Gradient-Based Optimizer: A New Metaheuristic Optimization Algorithm," *Information Sciences* 540 (2020): 131–159.

31. M. Mavrovouniotis, C. Li, and S. Yang, "A Survey of Swarm Intelligence for Dynamic Optimization: Algorithms and Applications," *Swarm and Evolutionary Computation* 33 (2017): 1–17.

32. G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and Profiling Scientific Workflows," *Future Generation Computer Systems* 29, no. 3 (2013): 682–692.

33. L. Zhang, M. Ai, K. Liu, J. Chen, and K. Li, "Reliability Enhancement Strategies for Workflow Scheduling Under Energy Consumption Constraints in Clouds," *IEEE Transactions on Sustainable Computing* 9 (2023): 155–169.

34. J. Yu and R. Buyya, "Scheduling Scientific Workflow Applications With Deadline and Budget Constraints Using Genetic Algorithms," *Scientific Programming* 14, no. 3–4 (2006): 217–230.

35. M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Transactions on Cloud Computing* 2, no. 2 (2014): 222–235.

36. N. Rizvi, D. Ramesh, L. Wang, and A. Basava, "A Workflow Scheduling Approach With Modified Fuzzy Adaptive Genetic Algorithm in IaaS Clouds," *IEEE Transactions on Services Computing* 16, no. 2 (2022): 872–885.

37. H. Qiu, X. Xia, Y. Li, and X. Deng, "A Dynamic Multipopulation Genetic Algorithm for Multiobjective Workflow Scheduling Based on the Longest Common Sequence," *Swarm and Evolutionary Computation* 78 (2023): 101291.