Journal of Network and Systems Management (2024) 32:89
https://doi.org/10.1007/s10922-024-09863-3

Tri-objective Optimization for Large-Scale Workflow Scheduling and Execution in Clouds

Huda Alrammah[1] · Yi Gu[1] · Daqing Yun[2] · Ning Zhang[3]

Received: 18 April 2024 / Revised: 14 August 2024 / Accepted: 15 August 2024

Abstract
Cloud computing has become the most popular distributed paradigm with massive computing resources and a large data storage capacity to run large-scale scientific workflow applications without the need to own any infrastructure. Scheduling workflows in a distributed system is a well-known NP-complete problem, which has become even more challenging with a dynamic and heterogeneous pool of resources in a cloud computing platform. The aim of this work is to design efficient and effective scheduling algorithms for multi-objective optimization of large-scale scientific workflows in cloud environments. We propose two novel genetic algorithm (GA)-based scheduling algorithms to assign workflow tasks to different cloud resources in order to simultaneously optimize makespan, monetary cost, and energy consumption. One is multi-objective optimization for makespan, cost and energy (MOMCE), which combines the strengths of two widely adopted solutions, genetic algorithm and particle swarm optimization, for multi-objective optimization problems. The other is pareto dominance for makespan, cost and energy (PDMCE), which is based on genetic algorithm and non-dominated solutions to achieve a better convergence and a uniform distribution of the approximate Pareto front. The proposed solutions are evaluated by an extensive set of different workflow applications and cloud environments, and compared with other existing methods in the literature to show the performance stability and superiority. We also conduct performance evaluation and comparison between MOMCE and PDMCE for different criteria.

Keywords Workflow scheduling · Multi-objective optimization · Pareto dominance · Makespan · Cost · Energy consumption

Extended author information available on the last page of the article

Published online: 06 September 2024

1 Introduction

Supercomputing technology has mainly changed the way of conducting basic and applied sciences from traditional laboratory-controlled experimental methodologies to modern computational paradigms, involving complex computational analyses and extreme-scale simulations of physical phenomena, computational biology, climatic changes, etc. Those scientific applications typically contain a great number of tasks with precedence constraints that need to be further analyzed and processed by geographically located users and scientists, and thus are beyond the capability of traditional standalone PCs. Traditionally, scientists run these scientific workflows using clusters and grid platforms. Many grid projects, such as Pegasus [1] and ASKALON [2], have designed workflow management systems to control and execute workflows on the grids. However, these systems are expensive and inconvenient to expand resources [3]. Cloud computing is the latest paradigm of these distributed systems and has emerged as a promising environment for execution of large-scale workflow applications. Although cloud computing has shown significant benefits to the IT industry and science experiments, the current technology suffers from many challenges that need to be carefully addressed, which calls for efficient and effective solutions to schedule and optimize the performance of workflow applications running in the clouds with multiple objectives. Workflow scheduling/optimization is a crucial issue in the cloud to achieve the best resource utilization from cloud's massive resource pool, meanwhile optimize the performance of a workflow execution under precedence constraints.

Scheduling frameworks need to address the issues derived from the cloud resource features in order to meet the application objectives. Another challenge that must be taken into account by scheduling algorithms is finding a trade-off between performance and other competitive requirements, such as reducing monetary cost, minimizing energy consumption, and delivering results as quickly as possible, which adds more complexity to the algorithm design in the heterogeneous cloud environments. Workflow scheduling with multiple conflicting objectives is a multi-objective optimization problem (MOP), where the optimal solution needs to consider trade-off between the objectives. A lot of existing works consider a single or bi-objective optimization problem where the main objective is to minimize the makespan of the workflow. It is desirable to formulate a workflow scheduling with MOP as cloud providers and users strive for different goals.

This paper proposes two scheduling algorithms to simultaneously minimize makespan, cost, and energy consumption of a workflow execution in a cloud while taking inter-task dependencies into consideration. The first algorithm, named multi-objective optimization for makespan, cost, and energy (MOMCE), combines the merits of genetic algorithm (GA) and particle swarm optimization (PSO) algorithms, and has more diversity in the search space and a better ability to converge towards the optimal solution. The second one, named pareto dominance for makespan, cost and energy (PDMCE), is implemented based on GA and non-dominated solutions to achieve optimum convergence and diversity of the Pareto front.

We consider a workflow application modeled as a directed acyclic graph (DAG) and a heterogeneous cloud platform, along with a set of continuous datasets that have to be processed by each task in a workflow. The pay-as-you-go cost model is considered based on hourly periods and related to our energy model. Moreover, we consider an initial boot time for a computer node as it needs some time to be available for executing users' tasks. We formulate mathematical models and conduct an in-depth investigation through workflow execution dynamics to ensure an accurate performance prediction.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 constructs cost models and formulates the optimization problem. Section 4 provides a brief introduction to the multi-objective optimization problem (MOP). The approaches are detailed in Sect. 5. Performance evaluations are discussed in Sect. 6. Section 7 concludes the work.

2 Related Work

Workflow scheduling in distributed systems has been widely studied over the years, starting from homogeneous clusters and multiprocessors with a limited set of resources [4, 5] to the most recent utility-based cloud computing with heterogeneous resources [6–13]. Many solutions have been proposed to solve the scheduling problem using traditional techniques including first come first serve (FCFS), shortest job first (SJF), Min-min, and round robin (RR) [14], etc., which are mostly used to schedule parallel tasks and maintain proper utilization of resources. They are simple and deterministic, but usually get stuck in a local optima [15]. Workflow scheduling problems in heterogeneous distributed environments are NP-complete and involve an extremely large search space. It is infeasible and computationally expensive to find an optimal solution within polynomial time. Heuristic and meta-heuristic techniques are used to find near-optimal solutions and investigate global optima.

Many workflow schedulers have used heuristic techniques to find near-optimal solutions and optimize different metrics such as a single objective as the makespan [8, 16, 17] or bi-objectives as makespan with cost [18–20] or makespan with energy consumption [21, 22]. These heuristic scheduling techniques are based on list-based scheduling. List-based scheduling is proper for the workflow of many tasks competing for a limited number of resources, which prioritizes the tasks and maps them based on their priorities. The second type of heuristic methods is cluster-based algorithms [23, 24] that assume an unbounded number of machines; therefore, they are mostly not feasible for practical use. The last type is the duplication-based algorithms [25–27], which mostly have a high time complexity. In general, list-based scheduling is a famous type which has been used through literature. The majority of algorithms of this type consider either mono-objective or bi-objective optimization techniques and consider other objectives as constraints. The disadvantage of them is that the computed solution depends on workflow characteristics as different workflow requires independent constraints, so any change in the tasks will require recomputing these constraints. Moreover, the user needs to have a knowledge of the workflow's and resources' characteristics

to specify right constraints, for example, the user might constrain the execution to $100Wh$, while the entire energy execution requires $150Wh$, which will lead to some failures [28]. Finally, in order to modify these algorithms to handle multiple objectives, the time to compute the scheduling solution may increase tremendously, so currently, many efforts move to the third scheduling algorithms category.

Meta-heuristic methods have become popular techniques in the last decade for multi-objective optimization problems (MOPs) to find near-optimal solutions, which deal with a massive search space compared with heuristic techniques. Recently, a number of nature-inspired meta-heuristic-based techniques, such as artificial bee colony (ABC), ant colony optimization (ACO), bat algorithm (BA), differential evolution (DE) [29], genetic algorithm (GA), particle swarm optimization (PSO), and simulated annealing (SA), have been applied to the task scheduling problem. Meta-heuristic scheduling approaches have an iteration over a population, which consists of the scheduling solutions converging to an optimal one, instead of producing a single final solution as in the previously mentioned methods. These schedulers consider a centralized architecture that optimizes conflicting objectives. They provide a static solution prior to the execution of the workflow and are suitable for offline or stochastic problem types.

In the last decade, different research works have adapted the GA to solve the workflow mapping problems [6, 30–37]. Zhu et al. [34] present an evolutionary algorithm EMOGA based on GA, which does not take resource acquisition delay nor energy consumption into account. Rehman et al. [35] also propose an algorithm MOGA based on GA, which optimizes makespan and cost under deadline and budget constraints, and also provides an energy-efficient solution. In [38], Talukder et al. propose a scheduling approach using multi-objective differential evolution (MODE), which gives a trade-off solution for time and cost. Most works have focused on optimizing two objectives, and lacked a comprehensive workflow scheduling solution that combines realistic mathematical modeling and cloud resource characteristics. The GA encoding scheme mostly represents task-to-resource mapping without considering task dependencies. Also, their genetic operations consider only the mapping string without performing any operations on the task order string. In addition, most of them combine the objectives functions into a scalar fitness function using weights, where the selected values of these weights are critical for an accurate solution.

Other works have adapted PSO method [9, 39–43]. In [40], Rodriguez et al. present a static cost minimization algorithm under the deadline constraints, which combines resource provisioning and task scheduling methods for executing workflows in IaaS clouds. The algorithm considers basic cloud features such as a pay-as-you-go model, heterogeneity and elasticity of the resources, VM performance variation, and boot time. Differential evolution (DE) is a parallel direct search method and widely used for solving complex MOPs [44]. There is another meta-heuristic to solve global optimization problems named symbiotic organisms search (SOS) [45], which is

based on the interactive behavior of organisms. In [46], Anwar et al. present a hybrid bio-inspired metaheuristic for multi-objective optimization (HBMMO) algorithm based on PEFT algorithm [47] and SOS algorithm.

Other solutions use the Pareto front computation to generate a set of trade-off schedules called Pareto front, which allows users to select the most appropriate solution they need [48–50]. Multi-objective heterogeneous earliest finish time (MOHEFT) algorithm [48] is an extension to HEFT algorithm that optimizes both makespan and cost for scheduling workflows in Amazon EC2. It calculates a set of intermediate solutions of workflow scheduling in each step for each task, instead of one produced by HEFT. In [49], Fard et al. propose a Multi-Objective List Scheduling algorithm to compute a set of Pareto-based solutions and present a four-objective case study including the makespan, economic cost, energy consumption, and reliability. Some objectives are specified as constraints with their distances to the dominant solutions being maximized, while others with their distances to the dominant solutions being minimized. These algorithms produce good results when the workflow size is small. Recently, some researchers have proposed hybrid multi-objective algorithms by combining the merits of two or more approaches [42, 46, 51]. However, most of them take a long time to converge due to the random allocation of tasks for their initial population.

Our main contributions of this work are that we consider a multi-objective optimization problem of makespan, cost, and energy consumption, develop realistic mathematical models for workflows and cloud resources, and propose two scheduling algorithms, MOMCE and PDMCE, to optimize workflow executions with three objectives simultaneously. Moreover, each algorithm has unique contributions as follows. MOMCE develops an algorithm that combines the characteristics of GA and PSO algorithms for better diversity in search space. PDMCE develops novel genetic operators for crossover and mutation to explore the search space more effectively. It applies non-domination sorting and crowding distance to rank the solutions and select the elite individuals for the next population, and obtains a fine Pareto front once the termination criteria are reached. It also applies a Fuzzy-based approach to select the best compromised solution among a set of obtained solutions.

3 Models and Problem Formulation

3.1 Workflow Model

We model a scientific workflow as a directed acyclic graph (DAG), $G_w = (V_w, E_w)$ where $V_w = \{t_1, t_2, ..., t_m\}$ is a set of m tasks and E_w is a set of directed edges. Each task is characterized by its size CL_{t_i}, which represents computational load expressed in million of instructions (MI) [19]. The dependency between two tasks is modeled as a directed edge $e_{ij}(1 \leq i, j \leq m)$, and data transfer size along an edge e_{ij} is z_{ij},

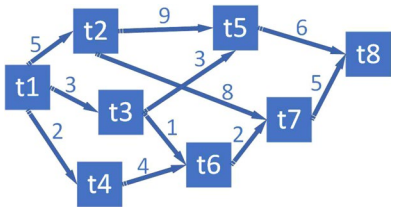


Fig. 1 An example of workflow application

representing the file size in Megabytes (MB). The precedence constraints ensure that a child task cannot start execution before receiving the data from all of its parents. We assume that a given DAG has one entry task t_{entry} with no predecessors and one exit task t_{exit} with no successors. For an illustration purpose, a sample workflow is shown in Fig. 1. The number above each edge is a data transfer size and the number inside a square is a task name, where the entry task t_{entry} is t_1 and the exit task t_{exit} is t_8, respectively.

3.2 Cloud Resource Model

We assume a hardware platform consisting of a set of heterogeneous resources modeled as a network graph, $G_{cn} = (V_{cn}, E_{cn})$, where $V_{cn} = \{r_1, r_2, ..., r_n\}$ denotes a set of n computer nodes connected by $|E_{cn}|$ links. Each node has its own configuration as represented by a three tuple $r_h = (p_{r_h}, c_{r_h}, f_{r_h})$, where p_{r_h} is the node processing capacity in terms of the number of instructions the CPU can process per second, million of instructions per second (MIPS) [19], c_{r_h} is the cost per billing period ($/hour), and f_{r_h} is the CPU frequency(MHz). The link l_{hk} between nodes r_h and r_k has bandwidth b_{hk} measured in megabits per second (Mbps) and minimum link delay (MLD) d_{hk}(seconds). MLD is a constant latency of each link during the transfer. When both tasks t_i and t_j are allocated to the same node, data transfer time becomes zero. We consider initial boot time D_{r_h}, provisioning delay, for a node as it needs some time to boot the system and be available for execution. It is assumed that there is no limitation on the number of resources used by a workflow application and leased from the provider. In addition, we adopt the pricing model used in commercial clouds based on a pay-as-you-go model. The number of used time intervals (per hour) is called billing period bp. Users are charged for each time interval even if they do not completely use the interval. For instance, for a unit of time that equals 60 min, if a resource is used for 70 min, the user will pay for two intervals of total 120 min. All nodes are located in the same data center, and internal data transfer is free of cost [52]. Although the cloud provider may charge for the storage services, they are not in the scope of this work. Additionally, each task is only mapped to one node and has to wait until all its required input data arrive. We assume that all nodes are processing on predefined configurations where the optimal CPU frequency is already calculated [53].

3.3 Time Model

Since there is no resource sharing for task execution and the allocated resources during the billing periods are exclusively used by the assigned task, the execution time of a task t_i running on a node r_h is calculated as: $T_{exec}(t_i, r_h) = \frac{CL_{t_i}}{p_{r_h}}$. No resource sharing also applies to the transfer time of a data size z_{ij} on an edge e_{ij} over a network link l_{hk} as: $T_{trans}(z_{ij}, e_{ij}, l_{hk}) = \frac{z_{ij}}{b_{hk}} + d_{hk}$. Each node has a start and a finish timestamps that should be updated during the mapping process. The start time of r_h denoted as $ST(r_h)$ is the earliest start time of the current mapped task, and the finish time $FT(r_h)$ is the last period used by the last mapped task t_l on r_h. Then, the Current Leased Period for a node $CLP(r_h) = [ST(r_h), FT(r_h)]$ is defined below:

$$CLP(r_h) = [ST(t_i, r_h), (ST(t_i, r_h) + (execCounter * bp))], \tag{1}$$

where *execCounter* is a counter of the execution of t_i, and increased by one for each billing period, denoted as: $execCounter = \lceil \frac{FT(t_l, r_h)}{bp} \rceil$, where $FT(t_l, r_h)$ is the finish time of t_l, and is defined next. We define the Start Time (ST) of task t_i on r_h in Eq. 2, and the Finish Time (FT) of task t_i in Eq. 3.

$$ST(t_i, r_h) = \begin{cases} D_{r_h}, & t_i = t_{entry} \\ \max\{T_{avail}(r_h), \max\limits_{t_p \in pre(t_i)} \{FT(t_p, r_s) \\ +T_{trans}(z_{pi}, e_{pi}, l_{sh})\}\}, & \text{otherwise} \end{cases} \tag{2}$$

$$FT(t_i, r_h) = D_{r_h} + ST(t_i, r_h) + T_{exec}(t_i, r_h). \tag{3}$$

In Eq. 2, r_s is the selected node of the parent task t_p, and $T_{avail}(r_h) = FT(t_i, r_h)$ is the time at which r_h is ready to execute a new task, which is dynamically changed during the operation. The provisioning delay D_{r_h} can be zero if $ST(t_i, r_h) < FT(r_h)$, which means a task can start at the remaining paid period of r_h and there is no waiting time to set up r_h. Otherwise, the execution will be delayed for the node to be ready. We consider the boot time of a node is equal to 97 s based on the results obtained by Mao and Murphy [54] for Amazon EC2 cloud [52]. Note that the actual start time and finish time of t_i on r_h can differ from the expected $ST(t_i, r_h)$ and $FT(t_i, r_h)$ due to performance variations. Makespan is the time from user submitting a workflow until receiving the output, which can be defined as the finish time of the last task:

$$Makespan_w = FT(t_{exit}). \tag{4}$$

3.4 Monetary Cost Model

Our pricing model considers commercial cloud systems using hourly billing periods based on Amazon EC2, in which partial hours are rounded up. The CPU frequency

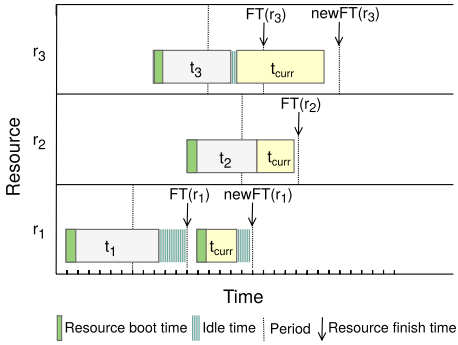


Fig. 2 Three scenarios of task execution

is already included in our execution model, which optimizes the speed of a resource to finish a task earlier, allowing the remaining paid period to be used efficiently. Therefore, when a task finishes with some remaining time in a paid period, we allow next task to use this leased resource to save costs.

Since there is no resource sharing, we have three scenarios for the usage time T_{usage} as shown in Eq. 5 and Fig. 2, where tasks t_1, t_2 and t_3 are already mapped. t_{curr} is currently being calculated for mapping, and it may fall in different scenarios illustrated by three nodes r_1, r_2 and r_3 as follow:

(i) On node r_1, the current task t_{curr} starts after the paid period ends, so the usage time equals to its execution time.
(ii) On node r_2, the usage time equals to zero as t_{curr} finishes before the end of an already paid billing period.
(iii) On node r_3, t_{curr} starts before the end of the paid period and ends after it. The time slot before $FT(r_h)$ is paid by the last mapped task on this node, which is t_3 in this case. So, the current task only needs to pay the remaining time slots, which is equal to the node's new finish time $newFT(r_3)$ minus the current finish time of the node $FT(r_3)$.

$$T_{usage}(t_i, r_h) = \begin{cases} FT(t_i, r_h) - ST(t_i, r_h) & FT(r_h) \leq ST(t_i, r_h) \\ 0 & FT(t_i, r_h) \leq FT(r_h) \\ newFT(r_h) - FT(r_h) & otherwise \end{cases} \tag{5}$$

Then, the monetary cost of executing task t_i assigned to node r_h is:

$$C_{exec}(t_i, r_h) = \lceil \frac{T_{usage}(t_i, r_h)}{bp} \rceil * c_{r_h}. \tag{6}$$

Thus, the total monetary cost of the workflow execution is:

$$Cost_w = \sum_{r_h \in V_{cn}} C_{exec}(t_i, r_h). \tag{7}$$

3.5 Energy Model

We focus on optimizing CPU power utilization for servers, and the idle servers can be put into sleep mode or switched off to save the total energy consumption [55]. The common option for energy management is to use dynamic voltage frequency scaling (DVFS) technique [56]. A standard DVFS enabled processor can execute a task in a discrete set of frequencies ($f_1 < f_2 < ... < f_n - 1 < f_n$). The digital complementary metal-oxide semiconductor (CMOS) circuit power consumption of a CPU consists of dynamic and static power consumption, denoted as $P = P_{dynamic} + P_{static}$. Static power consumption is proportional to the number of devices and includes the base power consumption of the CPU and other components. Dynamic power consumption has three components, represented by $P_{dynamic} = ACV^2f$, where A is the number of bits switching, C is the power-consumption capacitance, V is the supply voltage and f is the clock frequency. The dynamic power cost of CPU power consumption can be modeled as a convex function of the frequency $P_{r_h} = \delta_{r_h} + \alpha_{r_h} f_{r_h}^3$ [57], where δ_{r_h} is the constant power consumption, which is the static power consumed by CPU and other components such as disks, memory and I/O resources. $\alpha_{r_h} f_{r_h}^3$ is the variable power consumption, where α_{r_h} is a proportionality constant.

We consider the following power modes:

(i) Active: the power consumed during the execution, with the server operating under a discrete set of frequencies in the range of $[f_{h,min}, f_{h,max}]$, $P_{r_{h,active}} = P_{r_h}$.
(ii) Idle: the power consumed during the idle time between the executions of different tasks on the same server where it consumes only the base power, $P_{r_{h,idle}} = \delta_{r_h}$.

The energy consumption of node r_h in the active mode can be defined as:

$$E_{r_h,Active} = \sum_{i=1}^{k} T_{exec}(t_i, r_h) \times P_{r_{h,active}}, \tag{8}$$

where k is the number of tasks mapped to node r_h. The energy consumption of node r_h for its idle periods can be defined as

$$E_{r_h,Idle} = \sum_{idle=1}^{IDLE_{r_h}} T(r_{h,idle}) \times P_{r_{h,idle}}, \tag{9}$$

where $IDLE_{r_h}$ is a set of idling slots on node r_h, and $T(r_{h,idle})$ is the time duration of idling slot $idle$ on r_h. Therefore, the total energy consumption of node r_h is:

$$E_{r_h} = E_{r_h,Active} + E_{r_h,Idle}. \tag{10}$$

Based on the above formulations, the total Energy Consumption of the entire workflow execution is:

$$EC_w = \sum_{r_h \in V_{cn}} E_{r_h}. \tag{11}$$

3.6 Problem Formulation

Assuming a cloud provider has a data center which consists of a set of n heterogeneous computer nodes, and a user submits a request for a streaming application modeled as a DAG-structured workflow consisting of m tasks, the objective is to find a mapping scheme that assigns each task to a cloud node (without violating precedence constraints) to minimize the makespan, monetary cost, and energy consumption simultaneously. This can be formulated as a multi-objective optimization problem (MOP) as follows:

$$\min_{\text{all possible mappings}} \begin{matrix} Makespan_w, \\ Cost_w, \\ EC_w. \end{matrix} \tag{12}$$

A scheduling solution is represented as follows: $S = (R, M, Makespan_w, Cost_w, EC_w)$, where R represents the resource pool containing a set of active computer nodes, $R = \{r_h, ST(r_h), FT(r_h)\}$, with each node having a start time $ST(r_h)$ and a finish time $FT(r_h)$. M is the scheduling plan that determines a node r_h on which task t_i is mapped, with its expected start time $ST(t_i, r_h)$ and finish time $FT(t_i, r_h)$, and is represented by four tuples $M = (t_i, r_h, ST(t_i, r_h), FT(t_i, r_h))$.

4 Multi-objective Optimization: A Brief Overview

The process of simultaneously optimizing a collection of objectives is called multi-objective optimization problem (MOP) [58]. Assuming without loss of generality that minimization is the goal for all objectives, an MOP can be formally defined as:

$$\begin{aligned} &\text{Minimize} \quad F(x) = [f_1(x), f_2(x), ..., f_k(x)], \\ &\text{subject to} \quad g_j(x) \leq 0, j = 1, 2, ..., m \end{aligned} \tag{13}$$

where $k (k \geq 2)$ is the number of objective functions, and $x = (x_1, x_2, ..., x_n)$ is the vector representing the decision variables, $x \in S'$ where S' is the set of feasible solutions, and m is the number of constraints. The MOP of workflow scheduling in a cloud computing environment can be defined as finding all vectors $x = [x_1, x_2, ..., x_n]$ that minimize the vector $F(x) = [f_1(x), f_2(x), ..., f_k(x)]$. Each component of x represents a resource to which a task is mapped [28].

4.1 Design Components

Many MOP use meta-heuristic techniques to approximate the Pareto front based on the following search components.

1. Fitness Assignment: Guides the search algorithm toward Pareto optimal solutions for better convergence. A fitness function helps measure the quality of the solutions according to given optimization objectives. Some classifications of fitness assignment used by researchers in MOP are:

   (a) Scalar Approach: A more traditional approach used by many algorithms in the literature [19, 52], which transforms an MOP into a single-objective problem using techniques such as: weighted method, distance function, Min-Max method, or ϵ-constraint method.
   (b) Dominance-based Approach: Use the concept of dominance and Pareto optimality during the fitness assignment process. This approach can generate a set of Pareto optimal solutions in a single run without transforming the problem to a single-objective one. Instead of evaluating a single point at a time as in previous approaches, it assesses the quality of a solution using the information from previous iterations..

2. Diversity Preserving: Generate a diverse set of Pareto solutions in the objective space. Statistical density estimation methods can be used to estimate the density of the solutions, including the nearest-neighbor approach which uses the crowding distance concept. A higher crowding distance of a solution indicates that it is better and more diverse.
3. Elitism: A selection strategy where the best solutions (e.g., Pareto optimal solutions and best fitness values) found are chosen to build the next generation. It allows for fast and robust performance improvement of an MOP algorithm.

4.2 Pareto Solution and Dominance Concept

In a general MOP, it is hard to find an optimal solution for all objectives simultaneously due to the conflict criteria. Therefore, the desired solution is considered to be a set of potential solutions that are all optimal for one or more objectives [39, 49]. This set of all optimal solutions is called Pareto Optimal Set. We provide below a list of the concepts for Pareto solution theory [49] used in our algorithm.

1. Pareto Dominance: a solution x_1 dominates a solution x_2 if x_1 is at least as good for every objective as x_2, and x_1 is strictly better than x_2 in at least one objective, denoted by $x_1 \succ x_2$ and defined as:

   $$x_1 \succ x_2 \iff \forall if_i(x_1) \leq f_i(x_2) \wedge \exists jf_j(x_1) < f_j(x_2).$$

2. Pareto Optimality: a solution $x^* \in S$ is Pareto optimal if there are no other solutions Pareto-dominates it, and it is also called an efficient, or non-dominated solution, which is defined as:

   $$\nexists x \in S : x \succ x^*.$$

3. Pareto Optimal Set: is a set P^* of all the solutions that are not dominated by any member of the solution set, and it is defined as:

$$P^* = \{x^* \in S | \nexists x \in S, x \succ x^*\}.$$

4. Pareto Front: is the set of all Pareto efficient solutions, which is the line or the boundary of the Pareto optimal set in the objective space and can be defined as:

$$PF^* = \{F(x) = [f_1(x), f_2(x), ..., f_k(x)] | x \in P^*\}.$$

The goal of using the Pareto and dominance approach is to find a set of solutions that should be: (1) well-converged, meaning they should be as close as possible to the Pareto front by minimizing the distance to it, and (2) well-diversified, meaning they should be uniformly spread across the Pareto front and cover all possible ranges of the optimal solution.

5 Technical Solutions

We propose two algorithms to solve the MOP for scheduling a scientific workflow in a heterogeneous cloud environment: (1) One algorithm is based on Particle Swarm Optimization using the scalar approach for fitness assignment; (2) The other algorithm uses genetic operators and the dominance-based approach. These two algorithms are described in the following subsections.

5.1 MOP-Based Particle Swarm Optimization

We propose an algorithm based on a meta-heuristic optimization technique, named multi-objective optimization for makespan, cost and energy (MOMCE), which combines the merits of two popular MOP algorithms, genetic algorithm (GA) and particle swarm optimization (PSO). PSO is effective in achieving good results quickly but may sometimes get stuck in a local optima. GA gives better solutions due to its diversity in the search space, while it operates in a random-based manner, meaning that many of its steps are probabilistic which do not guarantee that new individuals outperform the previous ones. The proposed MOMCE solution leverages the strengths of both GA and PSO algorithms by implementing the following steps: (i) initializing a population, (ii) evaluating each particle using a fitness function, and (iii) repeating these steps for a predetermined number of iterations. The pseudo-code of the algorithm is presented in Algorithm 1, and discussed as follows.

Algorithm 1 MOMCE(G_w, G_{cn})

```
1: Randomly initialize all particle positions;
2: Replace one mapping by applying greedy approach;
3: Replace another mapping by applying HEFT algorithm;
4: Velocities = zeros, pBest position = particle's position;
5: Evaluate the fitness function;
6: gBest = the best particle among the swarm;
7: for G ← 1 to GNum do                          ▷ GNum: generations number
8:     if (G < GNum * crossProb) then
9:         sortParticlesSwarm(PSO_swarm);
10:        childId = (PNum/2) + 1;
11:        for p ← 1 to (PNum/2) do
12:            parent1 = particle[p];
13:            if (p == (PNum/2)) then
14:                parent2 = particle[1];
15:            else
16:                parent2 = particle[p + 1];
17:            Crossover(parent1, parent2, childId);
18:            childId++;
19:    Mutation(PSO_swarm);                       ▷ mutation operation
20:    for p ← 1 to PNum  do                      ▷ PSO main steps
21:        Calculate new velocity and position in Eq. 15;
22:        GetObjectives(particle[p].position, sortedTArray);
23:        Update resource pool (R) and mapping plan (M);
24:        Evaluate the fitness function in Eq. 14;
25:        Compare the fitness value with pBest and gBest;
26: Store the best solution, gBest, to the solution S;
27: return solution S;
```

1. Initialization: The first phase of the proposed optimization technique is generating an initial population of solution candidates. Here, a population is called a *swarm*, and each candidate solution is called a *particle*. The quality of the initial population is crucial as we need to reflect the heterogeneity of the resources which provides a guidance to the algorithm to explore different solutions and allows for faster convergence. All generated mapping schemes should be under the condition that they satisfy the precedence constraints. So we define *sortedTArray* which prioritizes tasks according to their computational requirements. One of the particles includes a mapping generated by a greedy approach which finds a feasible solution minimizing our objectives. Another particle includes a mapping generated by HEFT approach [17] which minimizes the makespan only. The remaining particles are initialized randomly, Lines (1–3). Each particle is characterized by three $1 \times m$ vectors, where m is the number of tasks: (i) Position (X_p) where $1 \leq p \leq PNum$, and $PNum$ is the swarm size, which encodes a mapping between tasks and resources. The range of the particle movement in (X_p) equals to n, the number of resources. (X_p) can be given as: $X_p = [x_1, x_2, x_3, ..., x_m]$. (ii) Velocity ($V_p$) defines the particle's movement, and all velocities are initialized to zero for all particles. (iii) Personal Best ($pBest_p$) records the best solution found so far by the particle, and its position is set to be equal to the initial particle's position, Line 4.

2. Fitness Function: At each generation, we evaluate particles based on the fitness function using their positions. The Cost Function for the current task t_{curr} to be mapped on resource r_h is denoted as $CF(t_{curr}, r_h)$. CF selects the best node that minimizes the following expression:

$$CF(t_{curr}, r_h) = \lambda_1 \times Makespan_{part}(G_w) + \lambda_2 \times Cost_{part}(G_w) + \lambda_3 \times EC_{part}(G_w), \quad (14)$$

where λ_1, λ_2 and λ_3 are fractional numbers ($0 \leq \lambda_{1,2,3} \leq 1$). Since three objectives have the same preference, we set $\lambda_1 = \lambda_2 = \lambda_3 = 1/3$. $Makespan_{part}(G_w)$, $Cost_{part}(G_w)$, and $EC_{part}(G_w)$ are the objectives that need to be evaluated during the partial mapping. $Makespan_{part}(G_w) = \sum_{t_i \in CP} FT(t_i, r_h)$, where CP is a critical path from t_{entry} to t_{curr}. $Cost_{part}(G_w) = \sum_{t_i \in M} C_{exec}(t_i, r_h)$, where t_i is the mapped task whose mapping result is stored in a tuple M. $EC_{part}(G_w) = \sum_{r_h \in R} E_{r_h}$, where r_h is the resource leased in the resource pool R.
3. Update pBest and gBest: Personal best (pBest) and global best (gBest) are essential elements in PSO algorithm as the progress of the particles is based on their values that change with each generation to ensure their movement towards the best solution. pBest is a vector for each particle that holds the best solution found so far by the particle, where gBest is one vector for the whole swarm holds the best solution among all particles. We compare the particle's current position with its best position from the previous generation and update pBest. For gBest, it is equal to the solution with the smallest fitness value among all.
4. Selection: Not all generated particles are evolved through the next operator. Therefore, The top-half best particles with the minimum fitness values are marked and regarded as (elite) parents for placing them in a mating pool for crossover operator. We sort the swarm in an ascending order according to their fitness. Then, select two particles as parents in Lines 9–16, and pass them to the crossover function.
5. Crossover: The occurrence of crossover is based on the crossover rate *crossProb*. We use a single-point crossover to generate a new child. A random number is selected in the range of the total number of tasks to indicate the division point. The child's position contains two parts which represent the resource mapping. The first part takes the first mapping from parent1 until the division point. The second part takes the solution starting from

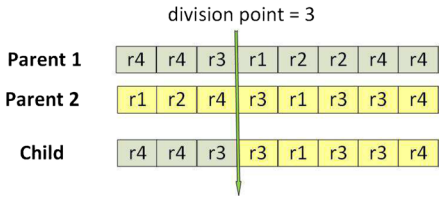


Fig. 3 An example of MOMCE crossover operator

this division point to the end of the parent2. The resulted child is saved to the swarm, and its fitness will be evaluated and saved. An example of the crossover operator is illustrated in Fig. 3, where a workflow of 8 tasks is mapped to 4 computing nodes and the division point is set to 3.

6. Mutation: The use of mutation operator is needed to avoid getting stuck in a local optima. We go through the particles, randomly select a task, and assign it to a random resource. If the mutated particle's fitness is better than the original one, save the mutated one. During this iteration, we need to check and update gBest.
7. Update Velocity and Position: The basic concept of PSO lies in accelerating each particle toward the best position (pBest) found by itself, and the global best position (gBest) obtained by any particle so far, with a random weighted acceleration at each step. Each particle is compared with its previous pBest. If they are equal, the velocity is decreased; otherwise, the velocity is increased. Similarly, the velocity is decreased when gBest and particle's position are equal; otherwise, it is increased. After that, the particle computes its new position by adding the velocity vector to the position vector to generate a new generation that has better fitness than the previous one. The new velocity and position are updated using Eq. 15. If the updated velocity and position exceed the search space, set them back.

$$\begin{aligned} V_p(t+1) = wv_p(t) + c_1 r_1(pBest_p(t) - X_p(t)) \\ +c_2 r_2(gBest(t) - X_p(t)), \\ X_p(t+1) = X_p(t) + V_p(t+1), \end{aligned} \tag{15}$$

where $V_p(t+1)$ is the velocity of particle p at iteration $t+1$, w is the inertia weight, c_1 and c_2 are acceleration coefficients, r_1 and r_2 are random numbers in the range [0,1], and $X_p(t+1)$ is the position of particle p at iteration $t+1$.

8. Termination Condition: The algorithm terminates when the maximum number of iterations is reached. Then, the smallest fitness saved in gBest is selected as the scheduling solution.

5.2 MOP-Based Non-dominated Genetic Algorithm

We present a multi-objective optimization algorithm based on pareto dominance for makespan, cost and energy (PDMCE) using GA and non-dominated solutions to achieve the required objectives. We further extend its operators to take both task scheduling order and task mapping into consideration. The key idea of the proposed PDMCE algorithm is illustrated below and the pseudo-code is presented in Algorithm 2.

Algorithm 2 PDMCE (G_w, G_{cn})

```
GNum : maximum number of generations;
Pop : population size;
mutProb : mutation probability;
crossProb : crossover probability;
GA_population: population contains Pop chromosomes;
EA : External Archive of size GNum;
//Initialization Phase
   One mapping saves minimum makespan schedule;
   Second mapping saves minimum cost schedule;
   Third mapping saves minimum energy schedule;
   Save B-rank metric to order string for the previous solutions;
   Rest mapping chromosomes are initialized randomly;
   Rest orders are initialized based on task's computational requirements;
   Evaluate the initial population;
   Save the non-dominated solution with maximum CD into EA;
//Main loop
for G ← 1 to GNum do
   for P ← 1 to Pop/2 do                                   ▷ Genetic operators
      parent1Id = TournamentSelection (population, -1);
      parent2Id = TournamentSelection (population, parent1Id);
      if (Math.random() ≤ crossProb) then
         child1 = Crossover(chrom[parent1Id], chrom[parent2Id]);
         child2 = Crossover(chrom[parent2Id], chrom[parent1Id]);
      if (Math.random() ≤ mutProb) then
         child1 = Mutation(child1);
         child2 = Mutation(child2);
      Evaluate children, and add them to childrenPopulation;
   Apply non-dominated sorting on parent and children populations;
   Get the CD for all individuals;
   GA_population = solutions in lower rank, and large CD for solutions in the same rank;
   Save one non-dominated solution with maximum CD into EA;
bestPopulation: non-dominated solutions from GA_population and EA;
return bestPopulation.
```

1. Encoding: The solution of the proposed optimization technique is called a *population*, and each candidate solution - individual, is called a *chromosome*. Each individual comprises of two components: scheduling order string and task mapping string. The scheduling order string represents a DAG structure under a topological sorting which satisfies the precedence constraints. We assign an integer index to each task according to their precedence. Note that if task t_i occurs before task t_j in the scheduling order, it does not necessarily mean the execution of t_i must start before t_j. The start time of a task is determined by its predecessors and the hosting node as shown in Eq. 2. The task mapping string indicates the mapping of a task to a computer node. The length of both strings equals to the number of workflow tasks.
2. Initialization: Assuming the size of a population is *Pop*, the initial population is generated by different methods in order to accelerate the search procedure for

faster convergence: (i) The first individual minimizes the makespan only by a greedy approach. (ii) The second individual finds the cheapest schedule using a greedy approach. (iii) The third individual minimizes the energy consumption only also by a greedy approach. In the above three strategies, we use B-rank metric [17] to compute the task order which indicates the distance of the task to the end of the workflow, and save it in the scheduling order string. (iv) For the last initialization strategy, we prioritize the tasks according to their computational requirements in each level. The initialization phase is shown in Lines 7–12, Algorithm 2.

3. Parent Selection: We use a tournament based selection approach to select the parent chromosomes for producing offspring in Lines 17–18, Algorithm 2. As shown in Algorithm 3, it takes the population and another parent id as inputs. The selection procedure randomly chooses an id from the population to include it in a mating pool. To ensure distinct parents, the id being -1 means this is the first parent for the parent pair, or it is not equal to the other parent id. Both parents should have different objective values for makespan, cost and energy in Line 6, Algorithm 3. Then, we need to get the non-dominated sort for the produced mating pool in Line 9, Algorithm 3. Finally, the algorithm returns a parent id by randomly choosing a non-dominated chromosome in the first front.
4. Crossover: We use a two-point crossover to generate two chromosomes (child1 and child2). The occurrence of the crossover is based on the crossover rate *crossProb* which equals 100%, as shown in Lines 19–21, Algorithm 2. The crossover operation defined in Algorithm 4 is applied to both the scheduling order and task mapping strings. As shown in Algorithm 5, two random points $p1$ and $p2$ are selected such that $1 \leq p1 < p2 \leq m$, where m is the total number of workflow tasks. We construct a *subOrder* and a *subMap* of size $(p2 - p1 + 1)$. *subOrder* holds the order for all tasks that fall in a range between $p1$ and $p2$ in the order string of parent1 and arranges them according to the execution order in parent2 under the dependency constraint. *subMap* holds the mapping of these tasks in *subOrder* as they appear in parent2. Further, copy these sub-strings to the new child string at index $(p1, .., p2)$. Then, copy all tasks' orders and mappings at index $(1, .., p1 - 1, .., p2 + 1, .., m)$ from parent1 to a new child. Finally, the algorithm will return the new child. For example, as shown in Fig. 4, we choose two random points: $p1 = 4$ and $p2 = 6$ from parent1. The tasks (t_4, t_7, t_5) between $p1$ and $p2$ take their execution order from parent2, which is (t_4, t_5, t_7), as well as their mapping (r_3, r_1, r_3) from parent2, and put them into a sub-schedule. Then, copy all tasks at indexes (1, 2, 3, 7, 8) with their orders and mappings from parent1 to child1. Finally, add the sub-schedule into child1 at indexes (4, 5, 6). We construct child2 in a similar way.
5. Mutation: Mutation operator is used to avoid getting stuck in a local optima. The topological level of each task can maintain the dependency constraint of the workflow tasks. We choose a random number between 0 and 1, and check whether it is less than or equal to mutation rate *mutPob*. If so, we perform mutation on both generated children from the crossover operation in Lines 22–24, Algorithm 2. As shown in Algorithm 6, a mutation that is done on the scheduling order will result in some changes in the task mapping. Therefore, if we cannot swap the tasks'

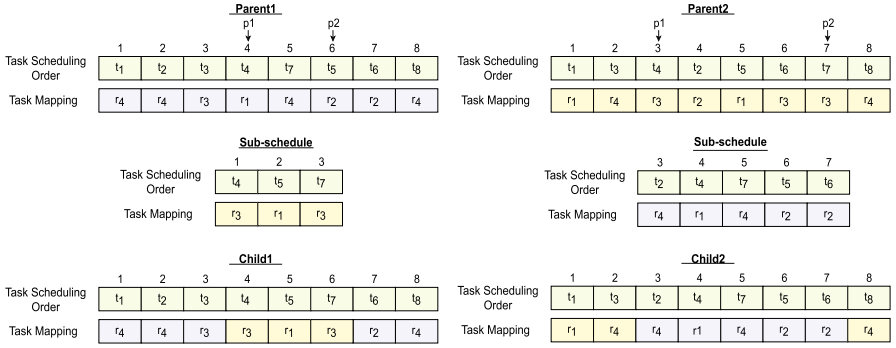


Fig. 4 An example of PDMCE with two point crossover

orders, we swap their mappings. The algorithm chooses two random numbers $p1$ and $p2$ such that $1 \leq p1 < p2 \leq m$, where m is the total number of workflow tasks. When the topological level of the tasks at $p1$ and $p2$ are equal, swap the tasks' orders in Lines 4–5, Algorithm 6. Then, if there are any task t_s, where $p1 < s < p2$, between the new swapped tasks such that $level(t_s) < level(t_{p1})$ and $t_{p1} \in succ(t_s)$, swap the tasks at $p1$ and s in Lines 6–9, Algorithm 6. Similarly, we check the dependency between t_s and the new task at index $p2$ in Lines 10–13, Algorithm 6. Otherwise, if we cannot swap the tasks' orders, we swap the task mapping at indexes $p1$ and $p2$ in Lines 14–15, Algorithm 6. Finally, the mutation procedure returns the new mutated child. An example to demonstrate the mutation operator is shown in Fig. 5, where the selected random points are $p1 = 4$ and $p2 = 7$, and $level(t_{p1}) = level(t_{p2})$. We swap the tasks t_5 and t_7, and check the tasks at the indexes $s = 5$ and $s = 6$, respectively. We find $level(t_3) < level(t_7)$, so we swap task t_3 with the task t_7 at index $p1 = 4$.

6. Fitness Evaluation (next generation selection): We use the concept of dominance and Pareto front [59]. We first combine the parent population with the children population, and each individual is set with a rank of non-dominated sorting. Each

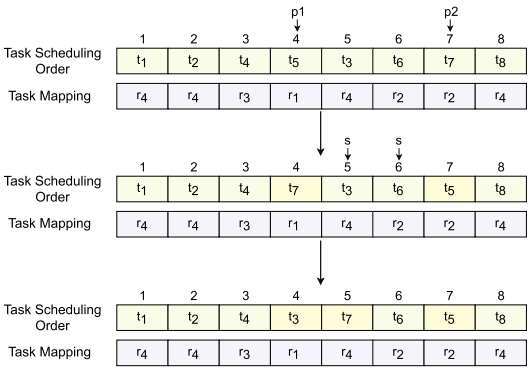


Fig. 5 An example of PDMCE mutation operator

solution with the same rank is placed in the same front. The smaller front solutions have a higher chance to maintain elitism to the next generation. Next, we go through the ranks and calculate the crowding distance (CD) to find the solution that maintains more diversity preserving. The selection method chooses the best (*Pop*) solutions from parent and children population of size ($2 * Pop$) as follows: (i) For the solutions in different ranks, the solutions with lower rank are preferred; (ii) For the solutions in the same rank, the solutions with larger crowding distance are preferred in Lines 26–28, Algorithm 2.

7. External Archive: The algorithm maintains an external archive (EA) to store one non-dominated chromosome found after the evaluation process in each generation in Line 29, Algorithm 2, based on the Pareto dominance and crowding distance discussed in step 6.
8. Termination Condition: The algorithm terminates once the maximum number of generations is reached. When the optimization process ends, the EA is combined with the last population, and the final set of all non-dominated solutions in the objective space is returned as the result, which is called Pareto front. For the particular objectives in this study, a candidate solution is a Pareto front if either it is at least as good as all other solutions for all three objectives in terms of makespan, cost and energy consumption, or it is better than all other solutions for at least one of these objectives.

Algorithm 3 TournamentSelection (population, otherId)

```
1: Pop : the size of the population;
2: s: Tournament size;
3: for i ← 1 to s do                                   ▷ fill up the mating pool
4:     while (True) do
5:         id = Math.random() * Pop;
6:         if (otherId == -1) || ((id ≠ otherId) && (chromosome(id).obj != chromosome(otherId).obj)) then
7:             matingPool[i] = chromosome(id);
8:             break;
9: getNonDominatedFronts(matingPool);
10: parentId = randomly select a chromosome id from the first front;
11: return parentId.
```

Algorithm 4 Crossover (parent1, parent2)

```
1: child1, child2: two new individuals;
2: child1 = twoPointCross(parent1, parent2);
3: child2 = twoPointCross(parent2, parent1);
4: return (child1, child2).
```

Algorithm 5 twoPointCross (parent1, parent2)

```
(p1, p2) = 1 ≤ p1 < p2 ≤ m;
subOrder: task order as in parent2;
subMap: mapping of subOrder tasks;
for i ← 1 to m do                                   ▷ construct the subOrder and suMap.
    for j ← p1 to p2 do
        if parent2.order[i] == parent1.order[j] then
            subOrder.push_back = parent2.order[i];
            subMap.push_back = parent2.mapping[i];
for i ← 1 to point1 − 1, point2 + 1 to m do                              ▷ produce the child.
    child.order[i] = parent1.order[i];
    child.mapping[i] = parent1.mapping[i];
x=1;
for i ← point1 to point2 do
    child.order[i] = subOrder[x];
    child.mapping[i] = subMap[x];
    x=x+1;
return (child)
```

Algorithm 6 Mutation (child)

```
(p1, p2) = 1 ≤ p1 < p2 ≤ m;
t_p1 = child.order[p1];
t_p2 = child.order[p2];
if (level(t_p1) == level(t_p2)) then
    swap(child.order[p1], child.order[p2]);                    ▷ 1. swap the task order.
    for s ← p1 + 1 to p2 do                                    ▷ 2. compare s with the new p1.
        t_s = child.order[s];
        if level(t_s) < level(t_p1)) && (t_p1 ∈ succ(t_s)) then
            swap(child.order[p1], child.order[s]);
    for s ← p2 − 1 to p1 do                                    ▷ 3. compare s with the new p2.
        t_s = child.order[s];
        if level(t_p2) < level(t_s)) && (t_s ∈ succ(t_p2)) then
            swap(child.order[s], child.order[p2]);
else                          ▷ 4. if we can't swap task order, we swap their mapping.
    swap(child.mapping[p1], child.mapping[p2]);
return (child).
```

6 Performance Evaluation

The proposed algorithms are implemented in C++ object-oriented framework. For a better performance evaluation and comparison purpose, we conduct the experiments in three categories summarized as below, and compare the performance with a set of existing approaches in the literature.

1. Performance Comparison between the proposed MOMCE and multi-objective discrete particle swarm optimization (MODPSO) [39].
2. Performance Comparison between the proposed PDMCE and other existing Multi-objective Optimization Problems.
3. Performance Comparison between MOMCE and PDMCE algorithms.

6.1 Experimental Settings

In order to conduct a thorough performance comparison, we develop a workflow and network generator program to create different sizes of workflows and networks by varying a four-tuple $(m, |E_w|, n, |E_{cn}|)$, where m is the number of tasks, $|E_w|$ is the number of edges, n is the number of nodes, and $|E_{cn}|$ is the number of links. All networks are considered fully connected. We conduct a performance comparison between MOMCE and MODPSO [39] algorithms, both of which optimize the same objectives. We select 20 test cases with different workflow sizes and divide them into four size groups, namely Small, Medium, Large, and XLarge, with five different workflow sizes in each group. For example, in the "Small" group, the five workflow sizes are (6,10), (10,18), (15,30), (19,36) and (22,44). In each pair, the first number indicates the number of tasks, and the second number represents the number of dependency edges in a workflow. For each workflow size, we generate 20 different workflow instances, and run each workflow instance on 4 cloud computer nodes using the same number of particle swarm $PNum = 20$ and the number of iterations $GNum = 30$. The average value of each objective for the 20 workflow instances is calculated in each test case using the same workflow size. The detailed information of the test cases and the cloud is tabulated in Table 1.

The performance comparison of the proposed PDMCE is done with a set of similar approaches, including MOHEFT [28, 48], MODPSO [39], and MOGA [35]. The test cases for the PDMCE algorithm are grouped into four categories: Small, Medium, Large and XLarge, varying from small to lager cases. For the illustration purpose, we list one case from each category as shown in Table 2. For example, a workflow in the "Small" category has 22 tasks interconnected with 44 edges, and is mapped on 4 cloud nodes where the population size is set to 60, and the number of generations (*GNum*) is set to 30. The sizes of the generated workflows vary in the following parameters within a suitably predefined range: (i) tasks' computational load from 1 to 500 MI; (ii) the inter-task data transfer size from 10 MB to 10 GB.

Table 1 20 MOMCE test cases in four different size groups

Group	Workflows (Task#, Edge#)	Cloud Nodes#	*PNum*	*GNum*
Small	(6,10), (10,18), (15,30), (19,36), (22,44)	4	20	30
Medium	(26,50), (30,62), (35,70), (40,78), (45,96)	8	50	50
Large	(50,102), (55,124), (60,240), (70,310), (80,420)	10	100	100
XLarge	(90,500), (100,660), (110,720), (120,810), (150,1100)	12	100	100

Table 2 PDMCE test categories

Category	Node#	Task#	Edge#	*Pop*	*GNum*
Small	4	22	44	60	30
Medium	8	50	102	60	50
Large	10	100	660	80	100
XLarge	12	150	1100	100	100

The generated workflows follow a common experimental settings used in the literature [60, 61].

We assume that the cloud data center provides 12 different types of nodes. The processing capacity and price are self-defined. The prices are in accordance with the processing capacity, which means that if the nodes have an ascending order of their processing capacity: $p_{r_1} < p_{r_2} < ... < p_{r_h} < p_{r_{h+1}}$, their prices have the same order: $c_{r_1} < c_{r_2} < ... < c_{r_h} < c_{r_{h+1}}$ [3]. Given the trade-off between energy consumption and performance, the optimal frequency level $f_{r_{h,opt}}$ for a node r_h is adopted from [57, 62], which should be within the discrete set $[f_{r_{h,min}}, f_{r_{h,max}}]$ and calculated as below:

$$f_{r_{h,opt}} = \sqrt[3]{\frac{\delta_{r_h}}{2\alpha_{r_h}}}.$$

Boot time of a node is set to 97 s based on the results obtained by Mao and Murphy [54] for Amazon EC2 cloud [52]. Table 3 summarizes the network characteristics used in our experiments.

For PDMCE, the crossover probability is set to 1.0, the mutation probability is set to 0.5, and tournament size is 7. For MOHEFT, the number of trade-off solutions equals to *Pop* as shown in Table 2. For Eq. 15, learning factors are defined as $c1 = c2 = 2.0$, inertia weight $w = 0.5$, and the random numbers for updating

Table 3 Network characteristics

Resource_h type	Capacity (MIPS)	Cost ($/h)	δ_i	α_i	f_{min}	f_{max}
1	100	0.10	60	4.0	0.5	1.0
2	150	0.20	75	5.0	0.8	1.5
3	200	0.35	65	4.5	1.0	2.0
4	250	0.46	75	4.0	1.3	2.5
5	300	0.58	90	4.4	1.5	3.0
6	350	0.73	90	4.5	1.8	3.5
7	400	0.90	105	4.4	2.0	4.0
8	450	1.05	105	6.5	2.3	4.5
9	500	1.45	110	7.0	2.5	5.0
10	550	2.02	120	7.5	2.8	5.5
11	600	2.50	150	7.0	3.0	6.0
12	650	3.00	150	7.8	3.3	6.5

velocity and position equations are $r1, r2 \in [0, 1]$. For MOGA, the crossover probability and the mutation probability are both defined as 0.5. To achieve the Pareto optimal solutions of the algorithms in comparison, each category is repeated 50 times, and the results are obtained by taking the average.

6.2 Performance Metrics

To evaluate the performance of the proposed solutions, we need to consider the following two things: (i) The distance between the optimal solutions obtained by the algorithms and the true Pareto front needs to be minimized. (ii) The distribution of optimal solutions should be uniform and equally spaced. Based on these two considerations, quality indicators are needed to evaluate the performance of the obtained solutions, in which, generational distance (GD) for the first one and Spacing for the second one [63] are adopted, respectively. We also measure the computational time needed to run each algorithm. We apply a Fuzzy-based approach [64] to select the best compromised solution among the set of solutions obtained by the algorithms.

In both indicators, we need to define the true Pareto front PF^*, which is obtained by merging the solutions calculated by all tested algorithms. The non-dominated solutions from this merged set are selected, and other solutions dominated by this true front are discarded. Then, we take the non-dominated results obtained by each algorithm, called the algorithm's Pareto front, for the comparison. Finally, we normalize the makespan, cost and energy consumption of the true Pareto front, and each algorithm's Pareto front of these objectives as they are on different scales.

1. Generational Distance (GD): This indicator measures the convergence of a given algorithm front and its proximity to the true Pareto front. GD is calculated as:

$$GD = \frac{(\sum_{i=1}^{F} d_i^2)^{\frac{1}{2}}}{F}, \tag{16}$$

   where F is the number of non-dominated solutions obtained by an algorithm, Pareto front, and d_i is the Euclidean distance in the objective space between each solution and the nearest member of the true Pareto front. The small value of GD reveals a better performance of the achieved solution set which indicates that the solution set is close to the true Pareto front.
2. Spacing: This indicator provides the information about the diversity of the solution set obtained by an algorithm. Spacing metric is defined as:

$$Spacing = \sqrt{\frac{1}{F}\sum_{i=1}^{F}(d_i - \overline{d})^2}, \tag{17}$$

   where d_i is the distance between two successive solutions in the obtained front by an algorithm, and $\overline{d}$ is the mean value of the distance measure d_i. The smaller

value of the spacing metric is desirable for evaluating the returned solution set, indicating the returned set has a uniform distribution.

3. Best Compromised Solution: The solution set obtained by PDMCE and the other tested algorithms comprises of the solutions that meet different objectives under certain conditions. Therefore, it is desirable to find the best compromised solution among the set, which can be used as another performance criteria to decide which algorithm produces the best result. The compromised solution can also be an option to a decision maker. We use the merged solution set of all algorithms through all executions without taking the non-dominated set. Then, we apply a Fuzzy-based approach [64] using a linear membership function defined as follows:

$$\mu_k = \begin{cases} 1 & f_k \leq f_k^{min} \\ \frac{f_k^{max} - f_k}{f_k^{max} - f_k^{min}} & f_k^{min} < f_k < f_k^{max} \\ 0 & f_k \geq f_k^{max} \end{cases}, \tag{18}$$

where μ_k is the membership value of the kth objective function, and f_k^{max} and f_k^{min} are the maximum and minimum values of this objective, respectively. The membership function μ^i for the ith non-dominated solution is defined as:

$$\mu^i = \sum_{k=1}^{K} \mu_k. \tag{19}$$

K is the number of objective functions, which is equal to 3 in this work. The best compromised solution is the one with the maximum value of the membership μ^i.

6.3 Comparison Between MOMCE and MODPSO

We implement multi-objective discrete particle swarm optimization (MODPSO) [39], which exploits the heterogeneity and the marketization of the cloud environment in order to find schedules that optimize makespan and cost using the PSO method. In addition, it utilizes the dynamic voltage and frequency scaling (DVFS) technique to minimize energy consumption. It initializes the voltage and frequency of each resource randomly, and apply the HEFT algorithm [17] to all particles for generating a feasible schedule that minimizes the makespan. However, it does not consider intervals for the monetary cost.

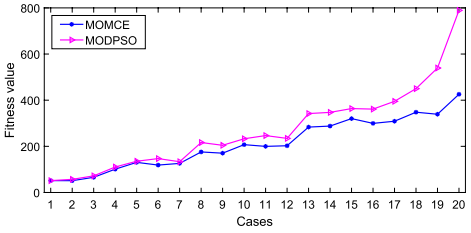
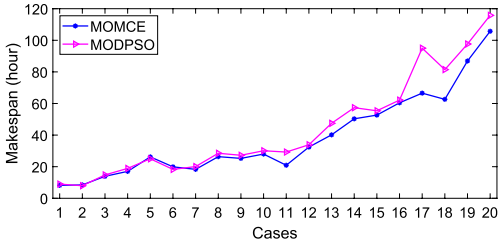
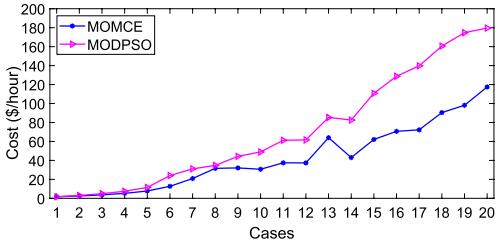


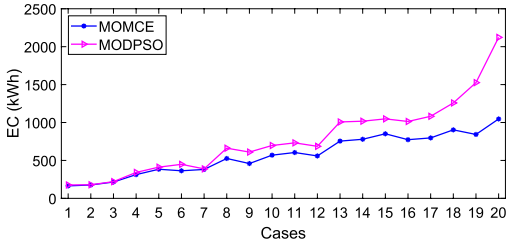
Fig. 6 The average fitness of MOMCE and MODPSO



(a) Average makespan.



(b) Average Cost.



(c) Average energy consumption.

Fig. 7 Average objectives of MOMCE and MODPSO

We observe that MOMCE achieves a better fitness value in all test cases and outperforms MODPSO in most of cases for each individual objective. Figures 6 and 7 plot the average values of 20 test cases. MOMCE achieves a better average fitness value at around 11.6% in “Medium" category (case 10), and 59.8% in “XLarge" category (case 20) than MODPSO in Fig. 6. In Fig. 7a, the average makespan of MODPSO is comparable to MOMCE, and in some cases, such as cases 5 and 6, MODPSO is better because it uses HEFT which only reduces the makespan to initialize their population. Therefore, most of the tasks are mapped to the fastest nodes without taking cost and energy into consideration. Even if MODPSO performs a little better (4.72% and 8.48%) on makespan in cases 5 and 6, MOMCE has significant improvements of the cost at around 39.71% and the energy consumption at around 61.66% in cases 5 and 6, respectively, as well as the overall fitness values. In case 20, MOMCE achieves a superior performance by 9.19% for makespan, 41.85% for cost, and 67.81% for energy consumption comparing to MODPSO.

In addition, our algorithm performs a fair distribution of the workload among the resources, while MODPSO has a low quality in terms of load balancing as there is more overloading on some fast nodes than others. It can be concluded that MOMCE outperforms MODPSO for better results and search efficiency.

To further investigate the robustness of MOMCE with other algorithms in comparison, we use 15 test cases out of the previous 20 cases to study the standard error of the algorithms. For each problem case, we randomly generate 10 problem instances and run each algorithm 40 times. We obtain the best fitness values for each problem instance and calculate the mean and the standard error as shown in Fig. 8. We observe some overlap between the two algorithms for small test cases, which indicates small search spaces make the convergence fast. With larger search space, MOMCE achieves a better performance with a smaller standard error. Increasing either the number of problem instances or the number of runs typically decreases the standard error of MOMCE furthermore. This figure has demonstrated the robustness and the stability of MOMCE in achieving a better fitness value in terms of optimizing three objectives simultaneously for various problem scales and topologies.

6.4 Comparison Between PDMCE and Other MOP Approaches

For a more general and extensive comparison, we implement the following algorithms.

- MOHEFT algorithm [17], which computes a set of Pareto-based schedules in order to optimize the makespan and cost. Another version of the algorithm is proposed to optimize the makespan and energy consumption in [28]. For each workflow task, MOHEFT builds a set of intermediate solutions based on available cloud nodes. Then, it measures the quality of these solutions using the dominance relationship, and ensures their diversity using the crowding distance method.
- MOGA is based on a basic GA to optimize the same objectives simultaneously. It works with a gap search algorithm to find the gaps between the consecutive busy intervals of the leased resources and fills them by independent tasks [35].
- MODPSO is described in Sect. 6.3.

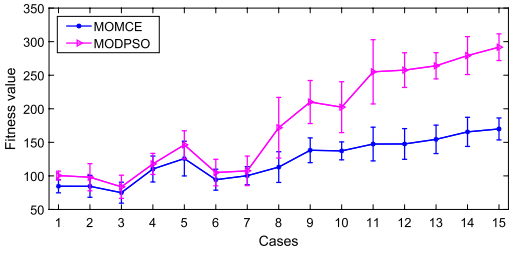


Fig. 8 Fitness values with standard errors for MOMCE and MODPSO

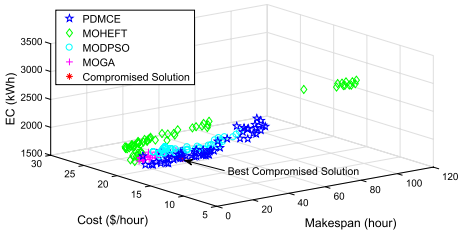


Fig. 9 Pareto fronts among four algorithms for Small category ($m = 22$)

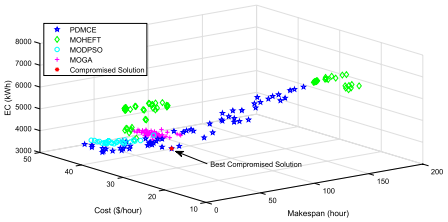


Fig. 10 Pareto fronts among four algorithms for Medium category ($m = 50$)

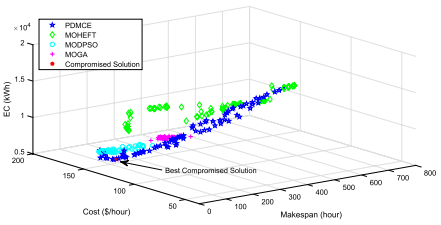


Fig. 11 Pareto fronts among four algorithms for Large category ($m = 100$)

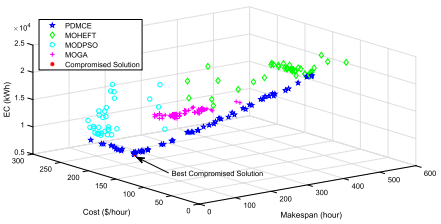


Fig. 12 Pareto fronts among four algorithms for XLarge category ($m = 150$)

All test cases are executed for 50 times using the PDMCE algorithm in comparison with three other algorithms, MOHEFT, MODPSO and MOGA. The obtained Pareto optimal solutions are plotted in Figs. 9, 10, 11, and 12, where we observe that a lower makespan with a faster computer node is usually associated with a higher cost. In addition, we see that most of the solutions produced by PDMCE are lying in a better front and better minimization region as compared to other algorithms for all test cases.

Figure 9 shows a workflow example of 22 tasks in the “Small" category in which all algorithms tend to compute optimal solutions easier and faster. Therefore, MODPSO results are comparable to ours, but PDMCE still achieves a better

performance and its Pareto front is superior. As the workflow sizes get larger, the process of reaching optimal solutions has become a challenging task. However, PDMCE still outperforms MODPSO with a better convergence and a uniform distribution of solutions. When comparing PDMCE with MOHEFT, the superiority is clear as MOHEFT makes more search directions in the solution space, while PDMCE avoids unnecessary search directions. MOGA produces close solutions to the minimization region, but it does not perform a uniform distribution of the solution set, while PDMCE delivers more consistent performance with a better distribution for non-dominated fronts. Moreover, when the user wants a good trade-off among time, cost and energy consumption, the Pareto front achieved by our PDMCE algorithm presents a large number of candidate solutions. We observe that the best compromised solution for each test case is obtained by the PDMCE algorithm as shown in the figures.

Table 4 presents the results of the performance metrics for the four algorithms. We generate 10 problem instances for each problem category, and run each algorithm 10 times. We then calculate the average of the GD and spacing along with standard error (SE) [42]. As the table shows, the GD value of the proposed PDMCE is smaller than other algorithms for all four cases, which indicates that PDMCE produces a better solution set and is closer to the true Pareto front because it uses the dominance sorting that provides a better convergence. The spacing metric also shows that PDMCE has lower values among others, which shows that our method has a better search efficiency due to the smooth distribution of its solutions as compared to those of MOHEFT, MODPSO and MOGA. Moreover, this result verifies that PDMCE is able to balance between objectives and diversity in the solution space.

Finally, the average runtime of each algorithm for the four cases is tabulated in Table 5. We notice that MOHEFT consumes more time comparing to other

Table 4 The performance metrics (in seconds) among four algorithms

Metric	PDMCE		MOHEFT		MODPSO		MOGA	
	Average	SE	Average	SE	Average	SE	Average	SE
Small								
GD	0.0048	0.0011	0.2303	0.0668	0.0302	0.0068	0.0182	0.0091
Spacing	0.0307	0.0041	0.1466	0.0975	0.0537	0.0136	0.0398	0.0091
Medium								
GD	0.0057	0.0009	0.1414	0.0430	0.0286	0.0126	0.0377	0.0122
Spacing	0.0221	0.0044	0.02842	0.1257	0.0324	0.0118	0.0342	0.0125
Large								
GD	0.0028	0.0008	0.1322	0.1232	0.0593	0.0266	0.1494	0.0451
Spacing	0.0206	0.0039	0.0952	0.0831	0.0423	0.0137	0.0306	0.0067
XLarge								
GD	0.0021	0.0018	0.4862	0.5091	0.0223	0.0122	0.1445	0.1039
Spacing	0.0261	0.0034	0.2361	0.2784	0.3002	0.4826	0.0367	0.0170

Table 5 Runtime (in seconds) comparison among four algorithms

Category	PDMCE	MOHEFT	MODPSO	MOGA
Small	0.5831	1.8082	0.687	0.5912
Medium	0.7541	2.952	1.3241	0.7425
Large	1.4771	5.427	2.4973	1.5458
XLarge	5.341	14.125	10.663	5.377

Table 6 MOMCE and PDMCE test categories

Node #	Task#	Edge#
4	(6, 10, 15, 19, 22)	(10, 18, 30, 36, 44)
8	(26, 30, 35, 40, 45)	(50, 62, 70, 78, 96)
10	(50, 55, 60, 70, 80)	(102, 124, 240, 310, 420)
12	(90, 100, 110, 120, 150)	(500, 660, 720, 810, 1100)

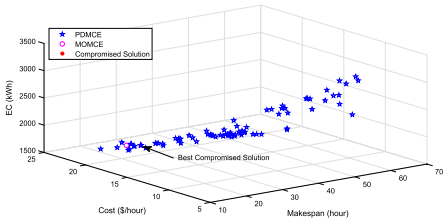


Fig. 13 Makespan, cost and energy consumption of PDMCE and MOMCE algorithms for workflow case 5 ($m = 22$)

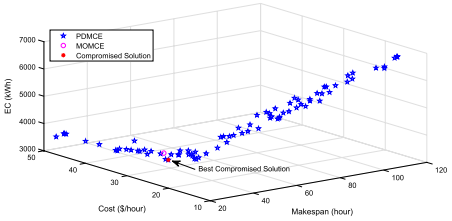


Fig. 14 Makespan, cost and energy consumption of PDMCE and MOMCE algorithms for workflow case 10 ($m = 45$)

algorithms, while PDMCE algorithm delivers the best performance among all four algorithms in comparison, with a smaller computational time.

6.5 Comparison between MOMCE and PDMCE

We conduct experiments using 20 different sizes of workflows from small to large categories as shown in Table 6 with 4, 8, 10, and 12 resource nodes, respectively. The node characteristics are specified in Table 3. For all test cases, the population sizes are set to 80, and the numbers of generations are set to 50. Each scheduling is executed 20 times, and the corresponding results are plotted in Figs. 13, 14, 15, and 16, respectively. For example, a workflow of 22 tasks mapped onto 4 computer nodes is selected from the first category in Table 6, and the scheduling results

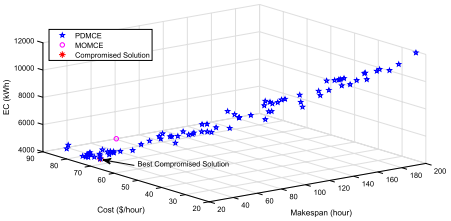


Fig. 15 Makespan, cost and energy consumption of PDMCE and MOMCE algorithms for workflow case 14 ($m = 70$)

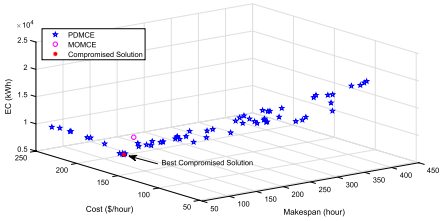


Fig. 16 Makespan, cost and energy consumption of PDMCE and MOMCE algorithms for workflow case 19 ($m = 120$)

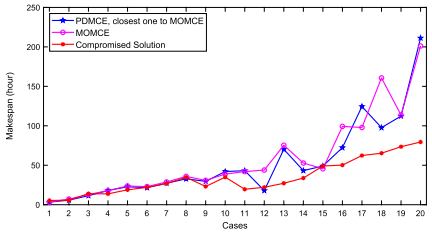


Fig. 17 The makespan values among PDMCE and MOMCE

obtained by both MOMCE and PDMCE are displayed in Fig. 13. In addition, we further compare both algorithms' solutions and compute the best compromised solution for each test case using Eq. 18.

We also observe that PDMCE algorithm delivers more consistent and better performance of varied solutions in finding better trade-off solutions for minimizing the makespan, cost and energy consumption among a set of solutions. Unlike MOMCE algorithm which produces one solution, PDMCE's results illustrate the basic multi-objective optimization procedure presented in Sect. 4. However, the solution obtained by MOMCE still falls within the minimization area and is close to the solutions obtained by PDMCE.

In order to conduct a fair comparison and analyze the effectiveness of both algorithms in terms of makespan, cost and energy consumption, we follow the similar methodology described in [39]. We compare the solution obtained by MOMCE algorithm with one solution in the Pareto set computed by PDMCE algorithm. We use 20 test cases, and for each one, we first run MOMCE algorithm 20 times and calculate the average values. Then, we run PDMCE algorithm to calculate a set of solutions in which we select the one that is the closest to MOMCE's value using Euclidean distance. Finally, the comparison is done in two ways to show the performance improvement of the objectives: (i) between MOMCE's solution and

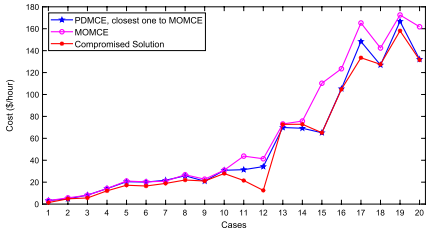


Fig. 18 The cost values among PDMCE and MOMCE

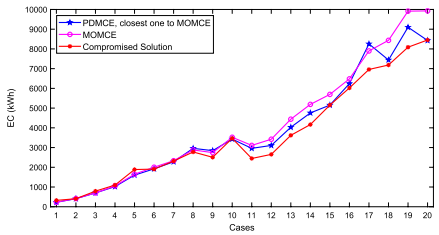


Fig. 19 The energy consumption values among PDMCE and MOMCE

Table 7 PDMCE improvement over MOMCE in four cases

Gain over MOMCE (%)				
Workflow (m, $\|E_w\|$)	Node#	Makespan	Cost	Energy
(22, 44)	4	4.21	3.08	2.03
(45, 96)	8	0	0.32	2.70
(70, 310)	10	19.93	9.02	8.71
(120, 810)	12	1.13	3.50	8.57

Table 8 Best compromised solution improvement over MOMCE in four cases

Gain over MOMCE (%)				
Workflow (m, $\|E_w\|$)	Node#	Makespan	Cost	Energy
(22, 44)	4	22.80	20.07	13.36
(45, 96)	8	10.37	10.43	1.60
(70, 310)	10	43.95	3.89	21.85
(120, 810)	12	42.86	8.72	20.30

PDMCE's closest solution, and (ii) between MOMCE's solution and the best compromised solution of the PDMCE algorithm. The obtained results for makespan, cost and energy consumption are plotted in Figs. 17, 18, and 19, respectively.

We observe that MOMCE is very close to at least one solution of PDMCE, while PDMCE consistently outperforms MOMCE in all the cases. For an illustration purpose, we tabulate the improvement of all four test cases in Table 7, where the results are reduced by 4.21% for the makespan, 3.08% for the cost, and 2.03% for the energy

Table 9 Runtime (in seconds) comparison of MOMCE and PDMCE

Workflow $(m, \lvert E_w \rvert)$	Node#	MOMCE	PDMCE
(22, 44)	4	0.032	0.313
(45, 96)	8	0.043	0.315
(70, 310)	10	0.366	0.623
(120, 810)	12	3.032	3.663

consumption in a small workflow of 22 tasks and 44 edges running on 4 computer nodes.

Table 8 shows the improvement of MOMCE and the best compromised solution. PDMCE can produce a better solution that minimizes 22.80% more of the makespan, 20.07% more of the cost and 13.36% more of the energy consumption for a small workflow of 22 tasks and 44 edges over MOMCE solution. The performance superiority of the best compromised solutions over MOMCE becomes more significant as the workflow size increases, which is reasonable in real-life scientific workflow applications. In a large workflow of 120 tasks and 810 edges, the performance of PDMCE is improved over MOMCE by about 42.86%, 8.72% and 20.30% for makespan, cost, and energy consumption, respectively. It is foreseeable that there will be even more performance improvements when the size of workflow applications further increases.

To further evaluate the robustness of the proposed methods, we measure the runtime of the MOMCE and PDMCE algorithms by executing both algorithms for 20 generations on a Windows 10 machine equipped with Intel(R) Core(TM) i5-3337U CPU of 1.80 GHz and 6.0 GB memory, and computing the average over 10 runs as shown in Table 9, where MOMCE converges with less computational time than PDMCE, as the evaluation process using non-dominated sorting in PDMCE causes the algorithm to be slower. However, the difference in runtime is not significant compared to the outstanding performance of PDMCE algorithm.

7 Conclusion

Cloud environment has pooled a variety of computing resources together with an on-demand access to meet various application needs and QoS requirements. However, the exponential growth of data size and the efficient deployment of scientific applications on those heterogeneous cloud environments have exposed new challenges to the scientific research community. In this work, we presented a multi-objective optimization problem in a cloud environment to minimize makespan, cost, and energy consumption of workflow execution, and proposed two workflow scheduling algorithms, MOMCE and PDMCE. The performance superiority of the proposed algorithms was demonstrated in an extensive set of comparisons with other existing algorithms in the literature. We have the following observations:

- MOMCE algorithm finds a good solution that handles a trade-off between the competitive goals with less computational overhead. When increasing the number of generations, we observe that MOMCE is getting better and closer to the best compromised solution.
- PDMCE exhibits a large number of non-dominated solutions which provide more flexibility for users to select a schedule that meets their needs. It may reach a global optima solution because the mechanisms of the initialization, selection, and evaluation are more intelligent.
- A Pareto front is an efficient tool that allows users to assess their preferences and select the most appropriate trade-off solution based on their QoS requirements.

It would be of our future interest to test the proposed algorithms in a real cloud environment using real-life scientific workflows. We would also like to take load balancing into consideration in our future work.

Author Contributions YG provided the research topic, guided the direction, discussed the main idea. HA did the experiments and collected the data. HA and YG wrote the main manuscript. DY and NZ participated in the discussions and provided insights into the solution finding and constructive comments on the paper writing and proofreading.

Data Availibility Data available upon request.

Declarations

Conflict of interest The authors declare no Conflict of interest.

References

1. Deelman, E., Singh, G., Su, M., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G., Good, J., Laity, A., Jacob, J., Katz, D.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci. Program. **13**(3), 219–237 (2005)
2. Fahringer, T., Prodan, R., Duan, R., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.L., Villazon, A., Wieczorek, M.: ASKALON: a grid application development and computing environment. In: The 6th IEEE/ACM International Workshop on Grid Computing, pp. 122–131 (2005)
3. Li, Z., Ge, J., Hu, H., Song, W., Luo, B.: Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds. IEEE Tran. Serv. Comput. **11**, 713–726 (2015)
4. Annie, S., Yu, H., Jin, S., Lin, K.C.: An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Trans. Parallel Distrib. Syst. **15**(9), 824–834 (2004)
5. Kwok, Y., Ahmad, I.: Dynamic critical-path scheduling: An effective technique for allocating task graph to multiprocessors. IEEE Trans. Parallel Distrib. Syst. **7**(5), 506–521 (1996)
6. Pirozmand, P., Hosseinabadi, A., Farrokhzad, M., Sadeghilalimi, M., Mirkamali, S., Slowik, A.: Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing. Neural Comput. Appl. **2021**, 1 (2021)
7. Mohammadzadeh, A., Masdari, M., Gharehchopogh, F.S.: Energy and cost-aware workflow scheduling in cloud computing data centers using a multiobjective optimization algorithm. J. Netw. Syst. Manag. **29**, 1–10 (2021)
8. Murad, S., Badeel, R., Alsandi, N., Faraj, R., Ahmed, R., Muhammed, A., Derahman, M., Salih, N.: Optimized min-min task scheduling algorithm for scientific workflows in a cloud environment. J. Theor. Appl. Info. Technol. **2022**, 480–506 (2022)

9. Farid, M., Latip, R., Hussin, M., Hamid, N.: Weighted-adaptive inertia strategy for multi-objective scheduling in multi-clouds. Comput. Mater. Contin. **72**, 1529–1560 (2022)
10. Behera, I., Sobhanayak, S.: Task scheduling optimization in heterogeneous cloud computing environments: a hybrid ga-gwo approach. J. Parallel Distrib. Comput. (2024). https://doi.org/10.1016/j.jpdc.2023.104766
11. Liu, B., Li, J., Lin, W., Bai, W., Li, P., Gao, Q.: K-PSO: An improved PSO-based container scheduling algorithm for big data applications. Int. J. Netw. Manag. **31**, e2092 (2020)
12. Arunagiri, R., Kandasamy, V.: Workflow scheduling in cloud environment using a novel metaheuristic optimization algorithm. Int. J. Commun. Syst. **34**, 4746 (2021)
13. Ma, X., Xu, H., Gao, H.: Real-time multiple-workflow scheduling in cloud environments. IEEE Trans. Netw. Serv. Manag. **18**, 4002 (2021)
14. Singh, S.: Performance optimization in gang scheduling in cloud computing. IOSR J. Comput. Eng. (IOSRJCE) **2**, 49–52 (2012)
15. Sharma, N., Tyagi, S.: A survey on heuristic approach for task scheduling in cloud computing. Int. J. Adv. Res. Comput. Sci. **8**(3), 1–4 (2017)
16. Gu, Y., Wu, Q., Rao, N.S.V.: Analyzing execution dynamics of scientific workflows for latency minimization in resource sharing environments. In: Proceedings of the 7th IEEE World Congress on Services, Washington DC, pp. 153–160 (2011)
17. Topcuouglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002). https://doi.org/10.1109/71.993206
18. Rodriguez, M., Buyya, R.: A responsive knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds. In: the 44th International Conference on Parallel Processing (ICPP), pp. 839–848 (2015)
19. Verma, A., Kaushal, S.: Cost-time efficient scheduling plan for executing workflows in the cloud. J. Grid Comput. **13**(4), 495–506 (2015)
20. Konjaang, J., Xu, L.: Multi-objective workflow optimization strategy (MOWOS) for cloud computing. J. Cloud Comput. **2021**, 1–19 (2021)
21. Tang, Z., Qi, L., Cheng, Z., Li, K., Khan, S., Li, K.: An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment. J. Grid Comput. **14**(1), 55–74 (2016). https://doi.org/10.1007/s10723-015-9334-y
22. Chen, H., Zhu, X., Qiu, D., Guo, H., Yang, L., Lu, P.: EONS: minimizing energy consumption for executing real-time workflows in virtualized cloud data centers. In: 45th International Conference on Parallel Processing Workshops, ICPP, pp. 385–392 (2016)
23. Boeres, C., Filho, J., Rebello, V.: A cluster-based strategy for scheduling task on heterogeneous processors. In: Proceedings of 16th Symposium on Computer Architecture and High Performance Computing, pp. 214–221 (2004)
24. Maurya, A.: Resource and task clustering based scheduling algorithm for workflow applications in cloud computing environment. In: International Conference on Parallel, Distributed and Grid Computing, pp. 566–570 (2020)
25. Bajaj, R., Agrawal, D.: Improving scheduling of tasks in a heterogeneous environment. IEEE Trans. Parallel Distrib. Syst. **15**(2), 107–118 (2004)
26. Lin, X., Wu, C.Q.: On scientific workflow scheduling in clouds under budget constraint. In: Proceedings of the 42nd International Conference on Para. Proc., pp. 90–99 (2013)
27. Bozdag, D., Catalyurek, U., Ozguner, F.: A task duplication based bottom-up scheduling algorithm for heterogeneous environments. In: Proceedings of the 20th International Conference on Parallel and Distributed Processing (2006)
28. Durillo, J., Nae, V., Prodan, R.: Multi-objective energy-efficient workflow scheduling using list-based heuristics. Futur. Gener. Comput. Syst. **36**, 221–236 (2014). https://doi.org/10.1016/j.future.2013.07.005
29. Abualigah, L., Diabat, A.: A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments. Clust. Comput. **2020**, 205–223 (2020)
30. Kumar, D., Sahoo, B., Mondal, B., Mandal, T.: A genetic algorithmic approach for energy efficient task consolidation in cloud computing. Int. J. Comput. Appl. **118**(2), 1–6 (2015). https://doi.org/10.5120/20714-3066
31. Leena, V.A., Beegom, A.S., Rajasree, M.S.: Genetic algorithm based bi-objective task scheduling in hybrid cloud platform. Int. J. Comput. Theo. Eng. **8**(1), 10 (2016)

32. Zhang, L., Li, K., Li, C., Li, K.: Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. Inf. Sci. **379**, 241–256 (2016)
33. Meena, J., Kumar, M., Vardhan, M.: Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint. IEEE Access **4**, 5065–5082 (2016)
34. Zhu, Z., Zhang, G., Li, M., Liu, X.: Evolutionary multi-objective workflow scheduling in cloud. IEEE Trans. Parallel Distrib. Syst. **27**(5), 1344–1357 (2016)
35. Rehman, A., Hussain, S., Rehman, Z., Zia, S.: Multi-objective approach of energy efficient workflow scheduling in cloud environments. Concurr. Comput. Pract. Exp. **34**, e4949 (2018)
36. Nagar, R., Gupta, D., Singh, R.: Time effective workflow scheduling using genetic algorithm in cloud computing. Int. J. Info. Technol. Comput. Sci. **10**, 68–75 (2018)
37. Ruan, F., Gu, R., Huang, T., Xue, S.: A big data placement method using nsga-iii in meteorological cloud platform. EURASIP J. Wireless Commun. Netw. **2019**, 143 (2019)
38. Talukder, A., Kirley, M., Buyya, R.: Multiobjective differential evolution for workflow execution on grids. Concurr. Comput.: Pract. Exp. **21**(13), 1742–1756 (2009)
39. Yassa, S., Chelouah, R., Kadima, H., Granado, B.: Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. Sci. World J. **2013**, 350934 (2013)
40. Rodriguez, M., Buyya, R.: Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans. Cloud Comput. **2**, 222–235 (2014)
41. Garg, R., Singh, A.: Multi-objective workflow grid scheduling using ϵ-fuzzy dominance sort based discrete particle swarm optimization. J. Supercomput. **68**(2), 709–732 (2014)
42. Verma, A., Kaushal, S.: A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. Parallel Comput. **62**, 1–19 (2017)
43. Beegom, A., Rajasree, M.: Integer-pso: a discrete PSO algorithm for task scheduling in cloud computing systems. Evol. Intell. **2019**, 227–239 (2019)
44. Storn, R., Price, K.: Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4), 341–359 (1997)
45. Cheng, M., Prayogo, D.: Symbiotic organisms search: a new metaheuristic optimization algorithm. Comput. Struct. **139**, 98–112 (2014)
46. Anwar, N., Deng, H.: A hybrid metaheuristic for multi-objective scientific workflow scheduling in a cloud environment. Appl. Sci. **8**(4), 13 (2018)
47. Arabnejad, H., Barbosa, J.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans. Parallel Distrib. Syst. **25**, 628–694 (2014)
48. Durillo, J., Prodan, R.: Multi-objective workflow scheduling in amazon ec2. Clust. Comput. **17**(2), 169–189 (2014)
49. Fard, H.M., Prodan, R., Barrionuevo, J.J.D., Fahringer, T.: A multi-objective approach for workflow scheduling in heterogeneous environments. In: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 300–309 (2012). https://doi.org/10.1109/CCGrid.2012.114
50. Zhou, X., Zhang, G., Sun, J., Zhou, J., Wei, T., Hu, S.: Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft. FGCS **93**, 278–289 (2019)
51. Manasrah, A., Ali, H.: Workflow scheduling using hybrid ga-pso algorithm in cloud computing. Wirel. Commun. Mobile Comput. **2018**(1), 1934784 (2018)
52. Ghasemzadeh, M., Arabnejad, H., Barbosa, J.: Deadline-budget constrained scheduling algorithm for scientific workflows in a cloud environment. In: 20th International Conference on Principles of Distributed System (OPODIS), pp. 1–16 (2017)
53. Alrammah, H., Gu, Y., Wu, C., Ju, S.: Scheduling for energy efficiency and throughput maximization in a faulty cloud environment. In: The International Conference on Parallel and Distributed Systems, pp. 561–569 (2017)
54. Mao, M., Humphrey, M.: A performance study on the VM startup time in the cloud. In: 2012 IEEE 5th International Conference on Cloud Computing, pp. 423–430 (2012)
55. Kliazovich, D., Bouvry, P., Khan, S.: DENS: data center energy-efficient network-aware scheduling. Clust. Comput. **16**(1), 65–75 (2013)
56. Chen, Y., Das, A., Qin, W., Sivasubramaniam, A., Wang, Q., Gautam, N.: Managing server energy and operational costs in hosting centers. Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 303–314 (2005)
57. Cao, F., Zhu, M., Wu, Q.: Energy-efficient resource management for scientific workflows in clouds. In: 2014 IEEE World Congress on Services, pp. 402–409 (2014) https://doi.org/10.1109/SERVICES.2014.76

58. Marler, R.T., Arora, J.S.: Survey of multi-objective optimization methods for engineering. Struct. Multidisc. Optim. **26**, 369–395 (2004). https://doi.org/10.1007/s00158-003-0368-6
59. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans. Evol. Comput. **6**, 182–197 (2002)
60. Wu, Q., Gu, Y.: Supporting distributed application workflows in heterogeneous computing environments. In: Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems, Melbourne, Australia, pp. 3–10 (2008)
61. Wu, Q., Gu, Y., Zhu, M., Rao, N.S.V.: Optimizing network performance of computing pipelines in distributed environments. In: Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium Miami, Florida (2008)
62. Alrammah, H., Gu, Y., Wu, C., Ju, S.: Scheduling for energy efficiency and throughput maximization in a faulty cloud environment. In: The International Conference on Parallel and Distributed Systems, pp. 561–569 (2017)
63. Deb, K., Jain, S.: Running performance metrics for evolutionary multi-objective optimization. In: Proceedings of Simulated Evolution and, Learning, pp. 13–20 (2002)
64. Abido, M.: Environmental/economic power dispatch using multiobjective evolutionary algorithms. IEEE Trans. Power Syst. **18**(4), 1529–1537 (2003)

Dr. Huda Alrammah is an assistant professor in the College of Computer and Information System at Prince Sultan University in Saudi Arabia, since August 2020. Prior to that, she graduated from Middle Tennessee State University (MTSU) as a Ph.D. student in December 2019. This research was initially conducted during her Ph.D. study under Dr. Yi Gu's direct supervision at MTSU.

Dr. Yi Gu is an associate professor in the Department of Computer Science at Middle Tennessee State University (MTSU). She received her M.S. and Ph.D. degrees in Computer Science from University of Memphis in 2008 and 2011, respectively, and the B.S. degree in Computer Science from Jiangsu University, P.R. China in 2005. She worked as an assistant professor of Computer Science at University of Tennessee Martin from 2011 to 2013, and then joined MTSU in 2013. Dr. Gu received her tenure in 2018. Her research interests include parallel and distributed computing, workflow scheduling and optimization, Cloud/Green computing, wireless sensor networks, and cyber security.

Dr. Daqing Yun is an Associate Professor in the Computer and Information Sciences (CIS) program at Harrisburg University (HU) of Science and Technology. His research interests include high-performance networking, distributed systems, and workflow optimization.

Dr. Ning Zhang is an Assistant Professor of Computer Science and Data Science in the Department of Mathematics and Computer Science at Fisk University. He earned his Ph.D. in Computational Science from Middle Tennessee State University, where he also completed an M.S. in Computer Science. Additionally, Dr. Zhang holds an M.S. in Signal and Information Processing from North China University of Technology and a B.S. in Electronic Engineering from Qingdao University, both in China. Before joining Fisk University in 2022, he served as an Assistant Professor of Computer Science at St. Ambrose University from 2019 to 2022. His research interests focus on Machine Learning, Data Mining, and Deep Learning, with an emphasis on supervised dimension reduction and neuroimaging data analysis.

Authors and Affiliations

Huda Alrammah[1] · Yi Gu[1] · Daqing Yun[2] · Ning Zhang[3]

✉ Yi Gu
Yi.Gu@mtsu.edu

Huda Alrammah
hka2f@mtmail.mtsu.edu

Daqing Yun
DYun@harrisburgu.edu

Ning Zhang
nzhang@fisk.edu

[1] Department of Computer Science, Middle Tennessee State University, Murfreesboro, TN 37132, USA

[2] Computer and Information Sciences, Harrisburg University of Science and Technology, Harrisburg, PA 17101, USA

[3] Department of Mathematics and Computer Science, Fisk University, Nashville, TN 37208, USA