



Bi-objective scheduling algorithm for scientific workflows on cloud computing platform with *makespan* and monetary cost minimization approach

Mirsaeid Hosseini Shirvani¹ · Reza Noorian Talouki¹

Received: 3 January 2021 / Accepted: 14 August 2021 / Published online: 15 November 2021
© The Author(s) 2021

Abstract

Scheduling of scientific workflows on hybrid cloud architecture, which contains private and public clouds, is a challenging task because schedulers should be aware of task inter-dependencies, underlying heterogeneity, cost diversity, and virtual machine (VM) variable configurations during the scheduling process. On the one side, reaching a minimum total execution time or *makespan* is a favorable issue for users whereas the cost of utilizing quicker VMs may lead to conflict with their budget on the other side. Existing works in the literature scarcely consider VM's monetary cost in the scheduling process but mainly focus on *makespan*. Therefore, in this paper, the problem of scientific workflow scheduling running on hybrid cloud architecture is formulated to a bi-objective optimization problem with *makespan* and monetary *cost* minimization viewpoint. To address this combinatorial discrete problem, this paper presents a hybrid bi-objective optimization based on simulated annealing and task duplication algorithms (BOSA-TDA) that exploits two important heuristics heterogeneous earliest finish time (HEFT) and duplication techniques to improve canonical SA. The extensive simulation results reported of running different well-known scientific workflows such as LIGO, SIPHT, Cybershake, Montage, and Epigenomics demonstrate that proposed BOSA-TDA has the amount of 12.5%, 14.5%, 17%, 13.5%, and 18.5% average improvement against other existing approaches in terms of *makespan*, monetary cost, *speed up*, *SLR*, and *efficiency* metrics, respectively.

Keywords Cloud computing · Scheduling · Meta-Heuristic Algorithms · Task duplication · Bi-objective optimization

Introduction

Recently, information technology (IT) was undergone a revolution. In this line, cloud computing attracted great attention in both industries and research communities for the sake of its pervasiveness, elasticity, and economy of scale [1]. Meanwhile, cloud computing is an amazing option for both individuals and organizations that do not have any exact resource usage pattern [2, 3]. For instance, in the case of the garment industry for Valentine's Day or Christmas, private cloud owners can exploit the public cloud to cover their sporadic burst of resource demand instead of proactively

resource procurement. Cloud has a wide range of applications from business to even academic projects. One of its abundant academic projects is in scientific workflow scheduling. Workflows include a set of tasks with different sizes, characteristics, and data dependency control flow between sub-tasks [4]; it is of comprehensive and complicated computation tool. Workflows such as LIGO [5, 6], SIPHT [5, 6], Epigenomics [5, 6], Cybershake [5, 6], etc., which are modeled in the form of directed acyclic graphs (DAGs), are popular paradigms in both industries and sciences [7]. Take that a university that has its private datacenter intends to execute such scientific workflows and requests more storage and computing resources during this process. Therefore, it can engage the public cloud to make hybrid cloud architecture. Note that the hybrid cloud is deemed a unique entity for users. One of the most important issues in the execution of workflows on cloud infrastructure is to schedule tasks and to allocate resources to these types of projects efficiently so that the maximum execution time of the last task, the so-called *makespan*, is minimized [8]. In this regard, schedulers

✉ Mirsaeid Hosseini Shirvani
mirsaeid_hosseini@iausari.ac.ir;
mirsaeid_hosseini@yahoo.com

Reza Noorian Talouki
reza.noorian@gmail.com

¹ Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran

encounter several challenges such as being aware of tasks inter-dependencies, underlying infrastructure heterogeneity, difference in VMs speed and pricing schemes, data transfer time on network channels, etc.

Several works have been published in the literature to solve workflow scheduling on cloud infrastructure with total execution time reduction, energy efficiency, reliability maximization viewpoints, but less paid attention to monetary costs which may have a big conflict with users' monetary budget. For instance, a load-aware heuristic strategy for dynamic workload and service scheduling in a cloud environment has been proposed by Lu et al. [9]. The main objective of the proposed algorithm was to improve execution performance; then, it has been validated by series of experiments. A novel hybrid discrete particle swarm optimization (HDPSO) algorithm was proposed to reduce maximum workflow execution time on cloud heterogeneous platforms [1]. To do so, this problem has been formulated into a single objective optimization problem. Although it had great improvement, it has not considered VMs' monetary costs. An energy-aware workflow scheduling algorithm has been presented with the aim of datacenter power management and keeping users' service level agreement (SLA) in [10]. Utrera et al. [11] have proposed an efficient algorithm to balance imbalance parallelizable programs on spare nodes with the aid of maximum resource utilization. Although it improves infrastructure resource utilization by packing tasks on the same processor, it does not consider user requirements as one of the most important stakeholders in the system. A multi-objective optimization workflow scheduling with execution time and energy efficiency has been propounded by Durilo et al. [12]. They applied the HEFT method as a list scheduler algorithm which has two important phases; at first, it constructs an ordered list of tasks guaranteeing topological order and dependency constraints; at the second phase, it picks a task with the highest priority to map on the processor which finishes the task execution at the quickest as possible time. It seems the suggested work is not suitable for users with tight budget constraints. Since workflows are modeled in the form of DAGs and there exist dependencies between tasks, data transfer between tasks worsens execution time, network traffic, and monetary costs as well. Therefore, the duplication technique may decrease network bandwidth usage and also can improve parallel path and degree of parallelism [13]. Specifically, it is a promising technique for communication-intensive DAGs which have a high communication-to-computation (CCR) rate. Qi Tang et al. have outlined task scheduling on a homogeneous platform by applying the duplication technique [14]. The outcome of their design was promising, but the duplication technique burdens more monetary costs as the scheduler must rent a couple of VMs instead of one. However, their algorithm did

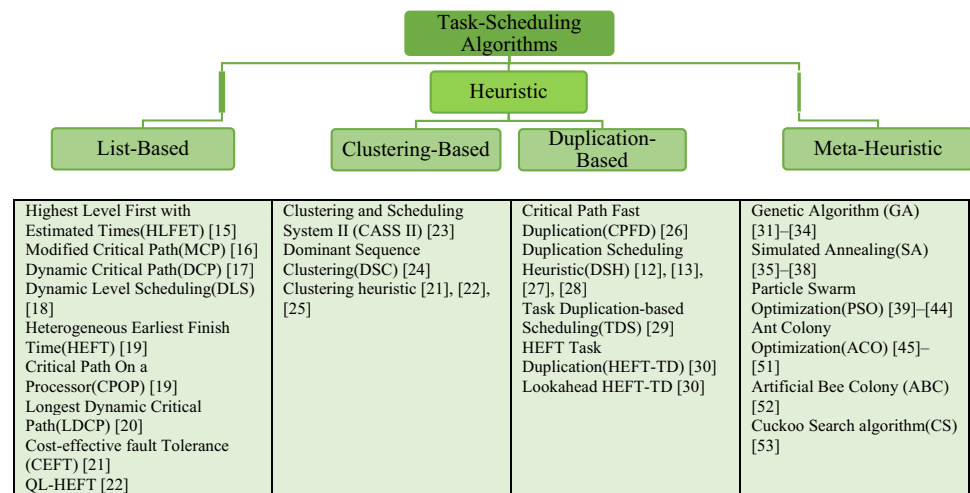
not take into account limitations for the number of allowable task duplications.

Reviews of published works in workflow scheduling on cloud platforms reveal that there exists a big gap in the literature for considering user monetary cost budget apart from the *makespan* minimization perspective. The most important innovation of the current paper which it conveys is that it formulates workflow scheduling problems on cloud platforms with *makespan* and monetary cost viewpoints. This is a bi-objective optimization problem under some constraints which is an NP-Hard problem. Since it is a discrete optimization problem, the simulated annealing (SA) algorithm is utilized which is very adaptive with discrete search space. However, canonical SA is a point-wise meta-heuristic computation; it cannot explore search space efficiently. This is the reason that a new population-based version of SA is presented; it is done by defining new operators and applying the crowding distance concept to make a bi-objective version of SA (BOSA). Also, to reach concrete results, the proposed algorithm is combined with the HEFT approach. In this way, it can explore search space efficiently to lead Pareto set of potentially conflicting objectives. The main contributions of the paper are as follows:

1. To present a new duplication-based list scheduler
2. To present a pricing model for VM deployment in cloud computing environment
3. To formalize workflow scheduling problem on heterogeneous cloud platforms into a bi-objective optimization problem with *makespan* and monetary cost optimization viewpoints
4. To present a bi-objective optimization based on simulated annealing task duplication scheduling algorithm (BOSA-TDA) along with new operators to solve the stated discrete bi-objective optimization problem

The rest of the paper is organized as follows. “[Related works](#)” classifies task scheduling algorithms in the form of related works. “[Problem background](#)” presents the problem background. Several proposed models are outlined in “[System, application, scheduling, and pricing models](#)”. “[Problem statement](#)” states problem formulation. Our proposed BOSA-TDA algorithm is elaborated in “[Proposed bi-objective optimization based on simulated annealing task duplication scheduling algorithm \(BOSA-TDA\)](#)”. To validate the current work, “[Simulation and evaluation](#)” is dedicated to simulation and evaluation. Finally, “[Conclusion and future work](#)” concludes this article along with future work inclination.

Fig. 1 Classification of Task Scheduling Algorithms in literature



Related works

Task scheduling is an important concept in all fields of computation domains especially when it is subjected to resource constraints; this is the reason that it has a long history. Figure 1 shows the categories of task scheduling algorithms in the literature.

A review in the literature reveals that the traditional task scheduling research has focused on list scheduling algorithms; chief amongst are heterogeneous earliest finish time (HEFT) [19] and critical path on a processor (CPOP) [19]. The basic idea of the HEFT version of list scheduling is that it consists of ordering a list of tasks by assigning priority to each one. The tasks are selected to the assigned priority and the ready task with the highest priority is removed from the task list to be assigned to an available virtual machine that guarantees the earliest finish time (EFT). In this category, another list scheduler, CPOP maps each task which is in the critical path, on the fastest VMs/processors in heterogeneous parallel platforms whereas the other tasks are mapped on VMs based on the EFT concept. Although the two aforementioned list schedulers were promising techniques in the primary era of scheduling, several improvements on these versions have been published in the literature to enhance schedulers' performance. In this regards, different extensions of list schedulers are such as CCP, CEFT, RHEFT, DHEFT, PEFT which have been customized for cloud environments [54–56]. For instance, robust HEFT (RHEFT) and distributed HEFT (DHEFT) have been developed to embed user's quality of service (QoS) requirements in the model apart from *makespan* [56]. A cost-effective fault-tolerance (CEFT) scheduling algorithm for real-time tasks in cloud environment was presented in [21]. This scheduling algorithm is applied in cloud environment with permanent or transient failure. The simulation result shows the CEFT gains

promising balance between low cost and deadline guarantee in real-time cloud systems. In addition to, a novel list scheduler algorithm which combines machine learning techniques and HEFT (known as QL-HEFT) was presented in [22]. The QL-HEFT scheduler utilizes upward ranking values from HEFT which are used for reward in Q-learning process. After an ordered list is provided; then, QL-HEFT engages the earliest finish time procedure to schedule high prior task which is placed in an ordered list by Q-learning process, on the fastest VM that returns the optimal result. The QL-HEFT was compared with three different classic list schedulers upward, downward, and CPOP approaches that were presented in [19]. The simulation results proved the superiority of QL-HEFT against counterparts in term of average response time. Also, Arabnejad and Barbosa in [54] have presented a novel list scheduler, predictable earliest finish time (PEFT) by introducing a look-ahead feature with computation of an optimistic cost table. It preserves time complexity against other existing approaches, but it lacks to consider rented VMs' cost. Some other famous list schedulers are: Highest Level First with Estimated Times (HLFET) [15], Modified Critical Path (MCP) [16], Dynamic Critical Path (DCP) [17], Dynamic Level Scheduling (DLS) [18], and Longest Dynamic Critical Path (LDCP) [20] in which their concentration are mostly on critical path management of given DAGs.

Recently, heuristic-based approaches became popular besides list schedulers. A heuristic-based algorithm normally finds a near-optimal solution in polynomial time. It searches for a path in the solution space at the expense of ignoring some possible trajectories [34]. Clustering and duplication are two prominent heuristic-based task schedulers in parallel systems [13, 25, 28–30, 57–62]. The heuristic-based scheduling algorithms can be classified as cluster-based schedulers such as [28, 57, 58] and task duplication-based schedulers

Table 1 Chronological comparison of emerging scheduling Algorithm in literature

Technique	Before 2012	2013–2016	2017–2021
Genetic algorithm (GA)	[33, 34]	[31, 32, 71]	[8, 65]
Simulated annealing (SA)	[35, 36]		[37, 38, 68, 72, 73]
Particle swarm optimization (PSO)	[39, 41–44]	[40]	[1]
Ant colony optimization (ACO)	[45, 47, 49, 66]	[46, 74]	[48, 51]
Artificial bee colony (ABC)		[52]	[67]
Cuckoo search (CS)		[53, 75]	
Clustering	[28, 57, 58]		[24]
Duplication	[28, 60, 61]	[12, 13]	[14, 30]
Fuzzy			[76]
Other approaches	[19, 26, 70]	[64]	[30, 69]

such as [13, 25, 29, 30, 59–62]. In the former method, the scheduler reduces communication costs by creating high communication-intensive dependent tasks as a cluster and mapping that clustered tasks on the same VM/processor whereas, in the latter approach, the duplication-based scheduler increases the degree of parallelism by executing a key subtask on more than one processor. Lin et al. [13] and Mishra et al. [28] have utilized clustering and duplication techniques in task scheduling problems. In [13], authors make some new graphs from input DAG by utilizing clustering, duplication, and replication methods. The main objective is to minimize *makespan* subject to keeping throughput and utilization at appropriate levels; then, one of the newly generated graphs which optimizes objective function and meets problem constraints is selected as a final solution. This work was validated in both real datasets and random task graphs which proved its superiority against some comparative algorithms. However, these heuristics are not appropriate in the platforms with a limited number of parallel VMs/processors [1, 8]. In addition, the duplication method was applied for task scheduling on homogeneous platforms in [14]. The outcome of their design was promising in *makespan* reduction, but their algorithm did not consider limitations for the number of allowable task duplications since it burdens users more monetary charge. Several heuristics have been devised to solve scheduling Bag-of-Tasks applications on hybrid clouds under due data constraints in [63]. This paper's trend was to optimize the total cost function which contains tardiness penalties and public cloud usage cost. Clustering and Scheduling System II (CASS II) has been presented to improve scheduling performance. To do so CASS II engages tasks on critical path to construct a cluster. Then, it assigns this cluster on the fastest available processor without considering any duplication technique [23]. Another duplication scheduling heuristic is discussed in an extended report published by Oregon state university [27].

Despite list schedulers and heuristic-based approaches which are biased deterministically, stochastic-search-based algorithms incorporate a combinatorial process in the search space for finding optimal solutions. Some stochastic-search-based algorithms such as meta-heuristic-based or even hybrid meta-heuristic-based approaches typically require sufficient sampling of candidate solutions in the search space and have shown robust performance on a variety of scheduling problems. In this regards, genetic algorithms (GAs) [8, 31–33, 64, 65] particle swarm optimization (PSO) [40, 43], Hybrid discrete PSO (HDPSO) [1], ant colony optimization (ACO) [47, 66], artificial bee colony algorithm (ABC) [52, 67] simulated annealing (SA) [35, 36, 68] cuckoo search algorithm (CS) [52, 93], the memetic discrete differential evolution algorithm [69] and tabu search (TS) [64, 70] have been successfully applied to different scheduling problems. Among them, GAs have been widely utilized to evolve solutions for many task scheduling problems [8, 31–33, 64, 65]. Table 1, chronologically, depicts the most cited related works in literature.

For instance, a shuffled genetic-based algorithm has been presented for task scheduling algorithms [8]. In its initial population, two individuals are filled by upward and downward ranking algorithms and the third individual is filled by level ranking which is drawn from the HEFT approach; then, the rest population is created by shuffling these individuals to produce feasible chromosomes. The same approach has been done by hybrid discrete particle swarm optimization (HDPSO) algorithm which produces initial swarm followed by two proved theorems [1]; then, it is randomly combined with the Hill Climbing method to make a good balance between exploration and exploitation in search space. Both presented models formulated scheduling problems as a single objective optimization by reducing the *makespan* viewpoint. A multi-objective optimization workflow scheduling with execution time and energy efficiency inclination has been

propounded by Durilo et al. [12]. Although this improves *makespan* and power consumption at the same time, it is not suitable for users with tight budget constraints. Another bi-objective optimization task scheduling with maximizing reliability and minimizing energy perspectives has been propounded by Zhang et al. [77]. This bi-objective HEFT (BOHEFT) scheduler weights system reliability more than performance metrics and maps tasks on heterogeneous VMs till low energy consumption and high reliability are simultaneously achieved. This algorithm ignores *makespan* and utilized VMs' cost taking into consideration. Since the task scheduling problem is a discrete optimization problem, the simulated annealing (SA) algorithm seems to be an efficient approach to reach the global optimum in discrete space [78]. Several versions of SA have been developed to figure out different scheduling problems [72, 73, 79]. As the SA algorithm is a point-wise optimization approach, it has two basic drawbacks. Firstly, it cannot explore search space efficiently in comparison with population-based evolutionary algorithms such as GA because it cannot generate a handful of candidate solutions. Secondly, it is hard to customize point-wise SA for multi-objective optimization problems. In [37] a hybrid genetic and simulated annealing algorithm (GASA) has been presented to solve scheduling problem in a cloud environment. This work is based on a list scheduler but to generate a handful of promising lists, it utilizes GA algorithm along with its strong crossover operator. To improve the gained solutions, it utilizes SA operators. Also, in [38] another hybrid genetic and thermodynamic simulated annealing algorithm (GATSA) was proposed to solve workflow scheduling in a cloud environment with regards to *makespan* minimization viewpoint. The proposed GATSA utilizes thermodynamic laws to gradually and variable decrease the temperature in the cooling phase. To this end, it applies variable cooling amount based on discrepancies fitness between each pair of consecutive solutions whereas it was neglected in the canonical SA. The conducted simulations in different circumstance proved the dominance of the GATSA against other counterparts in terms of scheduling evaluation metrics. In this line, a min-max ant colony optimization algorithm has been presented in literature to solve job scheduling in grid computing systems [50].

An overall review of the literatures associated with workflow scheduling reveals that there is a clear lack of workflow scheduling algorithms that optimizes both equally important *makespan* and monetary cost functions. In this line, the development of an intrinsic discrete-nature meta-heuristic algorithm such as SA can efficiently explore discrete search space. To solve the discrete bi-objective workflow scheduling problem, this paper extends a hybrid population-based bi-objective optimization algorithm based on simulated annealing and task duplication scheduling techniques in such a way that it can cover existing aforementioned shortcomings.

Problem background

In this section, a few concepts from the multi-objective optimization theory and canonical SA for a better understanding of this work are succinctly introduced.

Multi-objective optimization and crowding distance concepts

A multi-objective optimization problem is an issue that has several conflicting objectives which need to be optimized simultaneously. Without loss of generality, Eq. (1) outlines a multi-objective optimization problem with minimization inclination [80].

$$\text{minimize } F(x) = (F_1(x), F_2(x), \dots, F_k(x))^T. \quad (1)$$

An element $x^* \in X=(x_1, x_2, \dots, x_N)$ is called an N-dimensional feasible solution or a feasible decision. A vector $z^* := F(x^*) \in \mathbb{R}^k$ for a feasible solution x^* is called an objective vector or an outcome. In multi-objective optimization, there does not typically exist a feasible solution that minimizes all objective functions simultaneously. Therefore, attention is paid to non-dominated solutions (or Pareto optimal solutions); those solutions cannot be improved in any of the objectives without worsening at least one of the other objective. In mathematical terms, a feasible solution $X_1=(x_{11}, x_{12}, \dots, x_{1N}) \in X^N$ is said to dominate another solution $X_2=(x_{21}, x_{22}, \dots, x_{2N}) \in X^N$ as Eq. (2) indicates:

$$\{\forall i : F_i(X_1) \leq F_i(X_2) \text{ and } \exists j : F_j(X_1) < F_j(X_2) | i, j \in \{1 \dots k\}\}. \quad (2)$$

In the multi-objective domain, two concepts of convergence and diversity are very important issues where it differentiates other multi-objective optimization algorithms in term of performance. For instance, the famous NSGA-II algorithm introduces two effective selection criteria namely, Pareto non-dominated sorting and crowding distance to guide the search towards the optimal front [81]. The Pareto non-dominated sorting is used to divide the individuals into several ranked non-dominated fronts according to their dominance relations. The crowding distance is used to estimate the density of the individuals in a population; it is beneficial when the algorithm encounters memory size limitations. The multi-objective algorithm prefers two kinds of individuals: 1) the individuals with lower rank and 2) the individuals with larger crowding distance if their rank is the same [81]. For the last criterion, the crowding distance is employed that was defined in [82]. Our criterion is to prefer solutions with the lower rank and higher crowding distance value; the higher value of crowding distance means the solution sets were derived

from broader area. Finally, the non-dominated solutions are returned.

Simulated annealing (SA)

Simulated annealing is one of the most popular meta-heuristics developed that derives its inspiration from the natural world. In the case of simulated annealing, this inspiration comes from the behavior of fluids when they are subjected to control cooling such as in the production of large crystals. Simulated annealing for combinatorial optimization was introduced by Ref. [78] and independently by Ref. [83]. Despite other meta-heuristic approaches which have evolutionary trend, the SA tends to examine the worse solution apart from the good solution because in this way it runs away from getting stuck in a local optimal trap. During the annealing process, if a better solution is gained, it is accepted but in the case of producing a worse solution it is accepted by the amount of probability [84, 85]. It can be tuned in such a way that acceptance of the worse solution can happen at the beginning phase of the algorithm and when it reaches the end, the probability of worse solution acceptance is near to 0. In the other words, when SA became familiar with the search space, it behaves similar to other evolutionary algorithm at the last epochs; namely, it accepts only better solutions. The SA has one plus point and one negative point. The plus point is relevant to its flexibility for discrete optimization like task scheduling problems, but the negative point is relevant to its nature which is a point-wise algorithm. This is the reason the population-based of SA is presented to gain better results; as a matter of fact, the final results proved this idea.

System, application, scheduling, and pricing models

To present the proposed algorithm, some models are introduced for better understanding. Also, this paper presents mathematical optimization models. The nomenclature tabulated in Table 2 applied in the paper makes the paper easy to follow.

System model

In this section, understudy system model is introduced which executes DAG applications. The system contains a set of L different heterogeneous VMs which are interconnected with high-speed networks; namely, $VMset = \{VM_1^{Pr}, VM_2^{Pr}, \dots, VM_k^{Pr}, VM_{k+1}^{Pu}, \dots, VM_{k+m=L}^{Pu}\}$. In this model, the number of k VMs out of L makes a private cloud whereas the number of m VMs out of L makes a public cloud. Here, there is no difference between VMs of private and public clouds provided the underlying network is high speed. For the

sake of simplicity, it is taken a heterogeneous cloud platform with L different integrated VMs which is deemed a unique entity for users, but the heterogeneity is based on processors' architecture, speeds, and pricing schemes. For pricing model, the private cloud is considered on-demand whereas the public cloud is considered with charge period basis (c.f. Eqs. (17–21)). Moreover, the processing power in term of number of MIPS and monetary cost in term of \$/hour associated with each VM are variable and determined in advance. Such a system is depicted in Fig. 2.

In this model, the front-end layer is the user layer in which the user submits his/her request. The scheduling layer contains resource manager, job scheduler, DAG maker, and task scheduler which pays attention to the user's QoS request and his/her cost budget. In the back-end, there exists both private and public cloud that makes hybrid architecture. The main concentration of the current paper is on task scheduling in a hybrid architecture. For simplicity, a uniform high-speed network is considered in which all VMs can uniformly communicate with each other with the same bandwidth BW. So, for each $i, j = 1, \dots, L$ where $i \neq j$; $B(VM_i, VM_j) = BW$.

Application model

Each workflow application is modeled to directed acyclic graph (DAG); it is considered a directed acyclic task graph in $G = (T, E)$. Each vertex $v \in T = \{t_1, t_2, \dots, t_n\}$ in a DAG represents a task. Also, a DAG has two special nodes t_{entry} and t_{exit} that do not have predecessor and successor nodes respectively. The set of edges E in the graph where $\{e(t_i, t_j) \in E | t_i \neq t_j\}$ represents dependency between tasks t_i and t_j . That is a precedence constraint that indicates the task t_j can start its execution only after completion of the task t_i . The set $\{t_j \in T : e(t_j, t_i) \in E\}$ of all immediate predecessors of t_i is referred to as $Pred(t_i)$ and the set $\{t_j \in T : e(t_i, t_j) \in E\}$ of all immediate successors of t_i is referred to as $Succ(t_i)$. A task without any predecessor is called an entry task, i.e., $Pred(t_i) = \emptyset$ and a task without any successor is called an exit task, i.e., $Succ(t_i) = \emptyset$. The size of the task t_i and the weight assigned to edge $e(t_i, t_j)$ for computation and communication are represented by $Size(t_i)$ and $e(t_i, t_j)$ respectively. The amount of execution time for the task t_i on VM_k is calculated via Eq. (3). Besides, the average execution time of the task t_i on this heterogeneous platform is gained via Eq. (4). Where $ES(VM_k)$ and nP are execution speed of VM_k in terms of MIPS and the number of virtual processors in the system.

$$ET(t_i, vm_k) = \frac{Size(t_i)}{ES(VM_k)}, \quad (3)$$

$$\overline{ET}(t_i) = \frac{\sum_{\forall VM_k \in VMset} ET(t_i, VM_k)}{nP}. \quad (4)$$

Table 2 Nomenclature of used notations

Symbol	Meaning
T	Set of tasks in the application
E	Set of edges for precedence constraints among the tasks
nT	Number of tasks
nP	Number of Virtual machines in heterogeneous cloud system
t_i	The i th task in the application
TaskList	List of tasks in the first part of a chromosome
DuplicationList	List of duplication of tasks in the second part of a chromosome
$Size(t_i)$	Number of instruction in task t_i in term of Million Instructions (MIs)
γ	Duplication number for each task
t_i^γ	The γ th duplication of i th task in application
t_{entry}	The entry task with no predecessor
t_{exit}	The exit task with no successor
$Succ(t_i)$	The set of immediate successors of task t_i
$Pred(t_i)$	The set of immediate predecessors of task t_i
$Ranku(t_i)$	The upward rank of task t_i
$Rankd(t_i)$	The downward rank of task t_i
$e(t_i, t_j)$	A directed edge from task t_i to task t_j
$B(VM_k, VM_l)$	Bandwidth between virtual machines VM_k and VM_l
$TT(t_i, t_j)$	Transfer time in edge $e(t_i, t_j)$ where tasks t_i and t_j are running on different virtual machines VM_k and VM_l respectively via common bandwidth B
VM_k^β	The k -th VM in the virtual machine set with types $\beta = \{\text{public VM, private VM}\}$
$vmList$	A list of virtual machines and task features is provided in each virtual machine
$ES(VM_k)$	Execution speed of VM_k in term of Million Instruction Per Seconds (MIPS)
$EUnit(VM_k)$	Monetary cost for the execution of one unit of task on VM_k
$VM_k.List$	List of tasks assigned to VM_k
$BUT(VM_k)$	Boot-up time of VM_k (VM_k startup latency)
$ET(t_i, VM_k)$	Execution time for the task t_i on VM_k
$\overline{ET}(t_i)$	Average execution time of the task t_i on a cloud platform with heterogeneous VMs
$EC(t_i, VM_k)$	Execution cost for the task t_i on VM_k
$EST(t_i^\gamma, VM_k)$	The earliest start time of the task t_i on VM_k
$EFT(t_i^\gamma, VM_k)$	The earliest finish time of the task t_i on VM_k
$AST(t_i^\gamma)$	The actual start time of γ duplication of t_i task
$AFT(t_i^\gamma)$	The actual finish time of γ duplication of t_i task
$AUV(t_i^\gamma)$	The actual utilized virtual machine for execution of γ -th duplication associated to task t_i
$Makespan$	Total execution time
TEC	Total execution monetary cost
TTC	Total transfer monetary cost
ToC	Total monetary cost
CCR	Communication to computation ratio
SLR	Scheduling length ratio

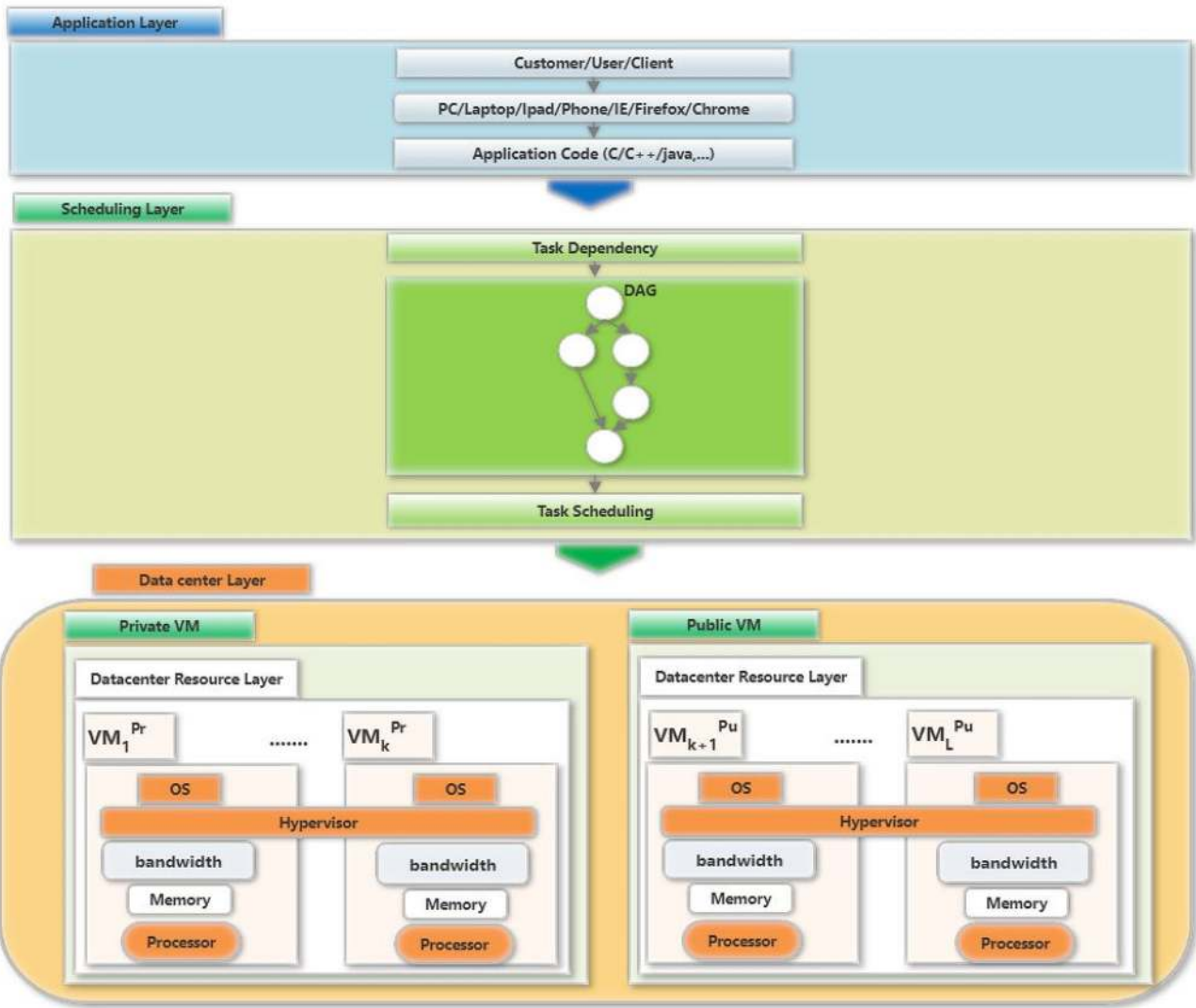


Fig. 2 Three-layer system model to execute dependent tasks on heterogeneous virtual machines

Moreover, the data transfer time (TT) between each pair of VMs can be calculated via Eq. (5) in which the amount of data being transferred and common bandwidth are effective in the TT parameter [86].

$$TT(t_i, t_j) = \begin{cases} 0 & \text{if } t_i \text{ and } t_j \\ & \text{are scheduled on the same VM} \\ \frac{e(t_i, t_j)}{BW} & \text{otherwise} \end{cases} \quad (5)$$

This algorithm is evaluated using synthetic data from five real-world scientific workflow applications, such as Montage (generation of image mosaics of the sky), Epigenomics (mapping of the epigenetic state of human cells), SIPHT (The bioinformatics project that is conducting a wide search for small untranslated RNAs in bacteria), CyberShake (generating seismic hazard maps for earthquake detection), and LIGO

(detection of gravitational waves in the universe) [5]. Figure 3a–e illustrate the aforementioned DAGs form projects.

Scheduling model and duplication technique

The scheduling model determines which task is assigned on which type of VM in regards to objective functions. The scheduling model that this paper applies is a list scheduler which has two important phases; at the first phase, it provides a list of ordered tasks with priority weight whereas at the next phase it picks a high priority task to assign on the available VM which guarantees the earliest finish time not the earliest start time. For the first phase, three approaches upward, downward, and level rankings of the HEFT algorithm are applied with the incorporation of duplication method if necessary. For the VM selection, two functions are engaged which are: Earliest Finish Time (EFT) and Earliest Start Time

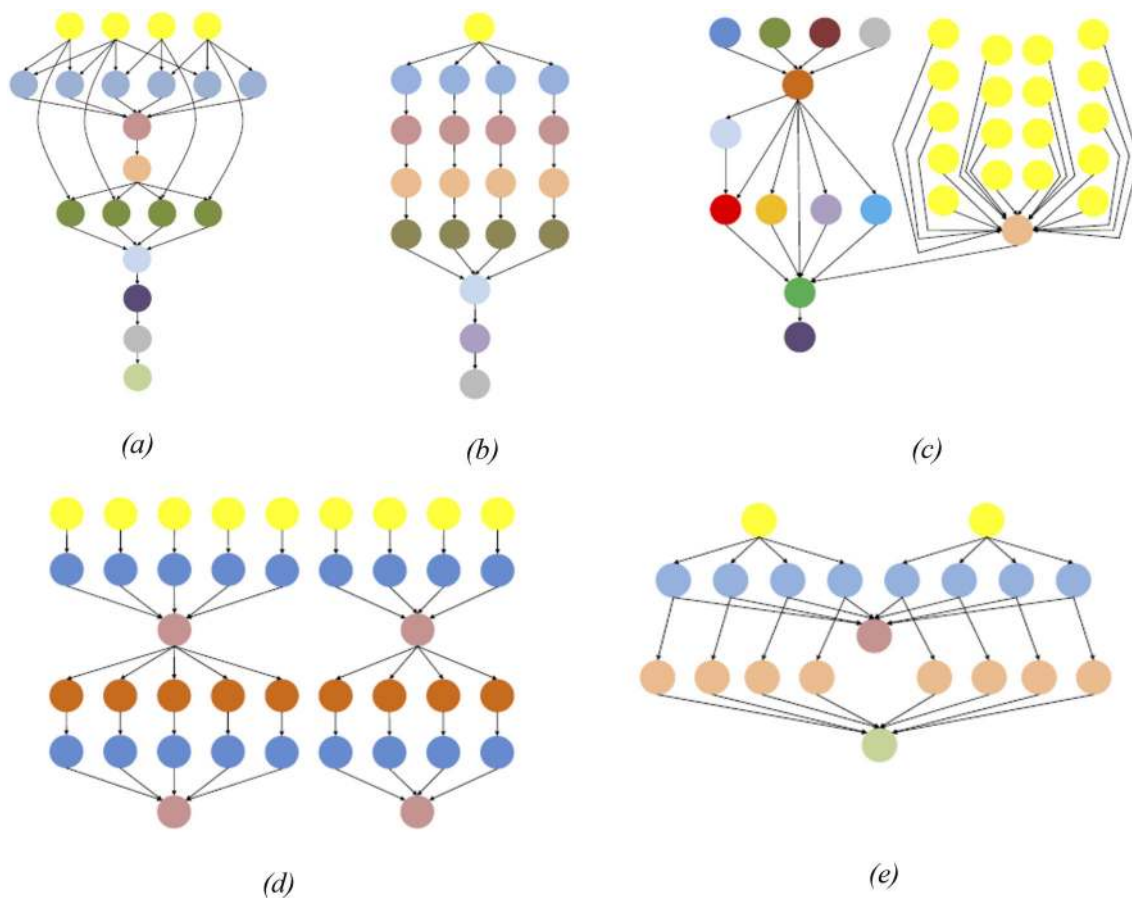


Fig. 3 Five real-world scientific workflow applications: a Montage [5], b Epigenomics [5], c SIPHT [5], d LIGO [5], and e Cyber Shake [5]

(EST). The first function indicates the earliest time in which a virtual machine $VM_k^{pr/pu}$ can finish the execution of the subtask t_i whereas the second indicates the earliest time that the execution can be started. To do so, two famous functions downward ranking $Rankd(.)$ and upward ranking $Ranku(.)$ are applied. The former starts from the *entry* task to the *exit* task to weigh each task a priority whereas the latter starts from the *exit* to the *entry* task. Both of them have recursive behavior. The downward ranking of a task t_i is recursively calculated by Eq. (6) where its value is considered zero for the entry task which Eq. (7) indicates.

$$Rankd(t_i) = \max_{t_j \in pred(t_i)} (Rankd(t_j) + \overline{ET}(t_j) + e(t_j, t_i)), \tag{6}$$

$$Rankd(t_{entry}) = 0, \tag{7}$$

where $pred(t_i)$ is the set of an immediate predecessor of task t_i and $e(t_j, t_i)$ is the average communication cost of edge $e(t_j, t_i)$, and $\overline{ET}(t_j)$ is the average computation cost of the task t_j . $Rankd(t_i)$ is the longest distance from the *entry* task to the task t_i , excluding the computation cost of the task

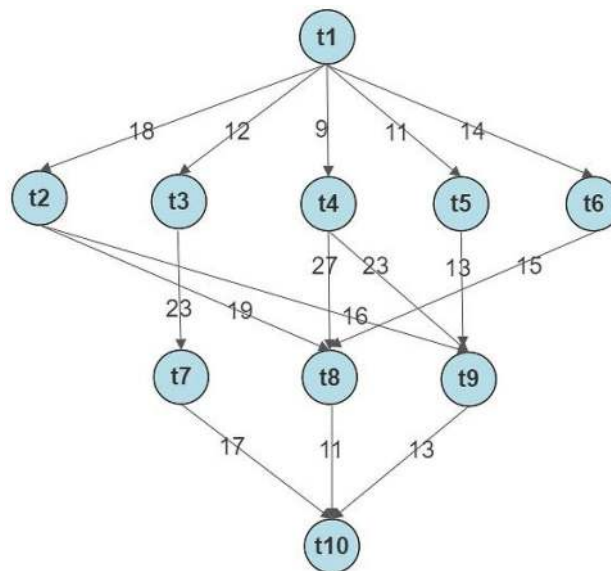


Fig. 4 An example of a DAG application with 10 tasks [97]

itself [19]. Similarly, the upward rank of a task t_i is recursively defined by Eq. (8) [19]. Also, for creating a sequence

Table 3 Available VMs, their execution times, and Monetary cost for the workflow depicted in Fig. 4

Tasks	$ET(t_i, VM_1)$	$ET(t_i, VM_2)$	$ET(t_i, VM_3)$	$\overline{ET}(t_i)$	$EC(t_i, VM_1)$	$EC(t_i, VM_2)$	$EC(t_i, VM_3)$	$AST(t_i^y)$	$AFT(t_i^y)$	$AUV(t_i^y)$	$Ranku(t_i)$	$Rankd(t_i)$
t_1	14	16	9	13.00	0.8	0.9	0.4	0	9	3	107.66	0.00
t_2	12	19	18	16.33	0.7	1.1	0.8	14	26	1	76.66	31.00
t_3	11	13	19	14.33	0.6	0.7	0.9	16	29	2	80.00	25.00
t_4	13	8	7	9.33	0.7	0.5	0.4	9	16	3	76.66	22.00
t_5	12	13	10	11.66	0.7	0.7	0.5	16	26	3	69.00	24.00
t_6	9	16	13	12.66	0.5	0.9	0.6	26	35	1	63.33	27.00
t_7	7	15	11	11.00	0.4	0.8	0.5	29	44	2	42.66	62.33
t_8	5	11	14	10.00	0.3	0.6	0.7	43	48	1	35.66	66.33
t_9	18	12	20	16.66	1.0	0.7	0.9	44	56	2	44.33	63.33
t_{10}	21	7	16	14.66	1.2	0.4	0.7	59	66	2	14.66	93

ordered of tasks based on upward ranking, each upward ranking value for each task t_i , except for *exit* task, is calculated via Eq. (8) whereas upward value for *exit* task is set by its average computation cost which Eq. (9) shows.

$$Ranku(t_i) = \overline{ET}(t_i) + \max_{t_j \in Succ(t_i)} (\overline{e}(t_i, t_j) + Ranku(t_j)), \tag{8}$$

$$Ranku(t_{exit}) = \overline{ET}(t_{exit}). \tag{9}$$

The term $Succ(t_i)$ is the set of immediate successors of task t_i . In addition, the term $Ranku(t_i)$ is the length of the critical path from task t_i to the *exit* task t_{exit} , including the computation cost of the task t_i . The third heuristic is the level ranking approach in which it assigns a level number to a task. The *entry* task has level 0, but for other tasks, the level is recursively calculated by Eq. (10). Then, the tasks are sorted based on their level ranking with increasing order.

$$Level(t_i) = \max_{t_j \in Pred(t_i)} (Level(t_j)) + 1. \tag{10}$$

In addition, in this paper, task scheduling performance is improved by utilizing the duplication technique which doubles critical tasks on different VMs because this technique enhances running parallelism. On the other hand, duplication may shorten the time interval in which VMs are at service; therefore, it can potentially decline monetary cost. Note that, duplication technique intrinsically leads to charge for a couple of VMs rent instead of one VM, so, there exists a clear conflict between *makespan* reduction and monetary costs. This is the reason this issue is formulated as a bi-objective optimization problem by applying the dominance concept for finding solutions that compromise between objectives. To apply the duplication technique to an ordered list, a random number is dedicated to each task t_i which has the minimum value, $min = 1$ and the maximum value, $max = \min \{count\{VMs\}, \max\{count(Pred\{t_i\}), count(Succ\{t_i\})\}\}$; these numbers are determined based on the number of VMs, the number of predecessors, and the number of successors associated with each task. Then, it can be balanced in the proposed enhanced simulated annealing process. After an ordered list is prepared by any ranking heuristic; then, the duplication technique is applied. Hereafter, each duplicated task is treated the same as the original task in the list. To apply the VM selection phase, new $EST(t_i, VM_k)$ is defined which is used to show the last time the task t_i whether original or duplicated task can wait for execution on VM_k . If t_i is an *entry* task in a DAG or it is the first task must be assigned in the VM_k 's task list that the $VM_k.List = \emptyset$ shows the case, the $EST(t_i, VM_k)$ is equal to the boot-up time of VM_k , shown by $BUT(VM_k)$; otherwise, the term $EST(t_i, VM_k)$ is calculated with Eq. (11). In this equation, t_j^y is y -th duplication of task t_j that is a member of the predecessor of t_i , otherwise task t_j is an original predecessor

< Task List :> (Ranku/Rankd/Level/Rand) list										< Duplication List :> (Mutation/Crossover/Rand) for each task $t_1: t_n$									
$t_?$	$t_?$	$t_?$...						$t_?$	t_1^y	t_2^y	t_3^y	...						t_n^y
t_1	t_2	t_4	t_6	t_3	t_5	t_7	t_9	t_8	t_{10}	2	1	1	2	1	1	1	1	1	1

Fig. 5 Task encoding model chromosome

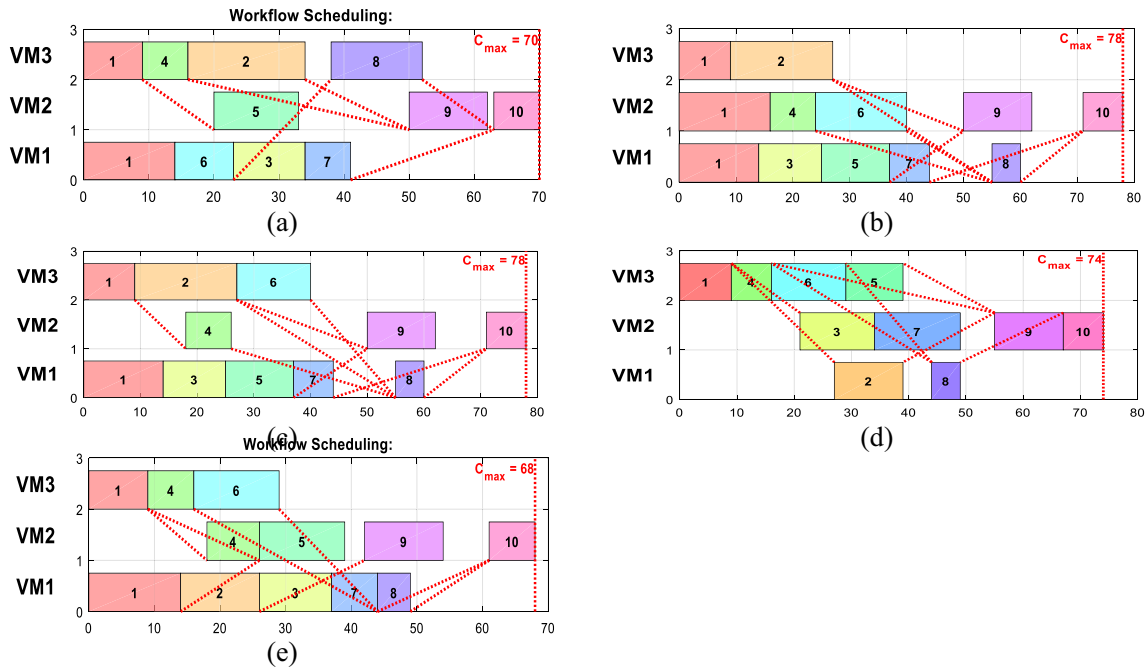


Fig. 6 Output of illustrative example with a NSGAI, b HEFT-TD, c Lookahead HEFT-TD, d SOGA, and e BOSA-TDA algorithm

of task t_i . The important note is that, the fastest duplicated task and the slowest original task in the predecessor of task t_i should be taken into consideration. In addition to, the function Avail (VM_k) is used to indicate the time which this VM's last task has been finished and it is available for the new task.

$$EST(t_i, VM_k) = \max \left\{ \text{Avail}(VM_k), \left\{ \max_{\forall t_j \in \text{pred}(t_i)} \{ \text{AFT}(t_j) + \text{TT}(t_j, t_i) \}, \min_{\forall t_j^y \in \text{duplicat}(t_i)} \{ \text{AFT}(t_j^y) + \text{TT}(t_j^y, t_i) \} \right\} \right\}. \tag{11}$$

When a new virtual machine VM_k is intended to be started before the task scheduling can be performed, it is needed to boot up the virtual machine VM_k in the system; where the function BUT(VM_k) is considered to measure this boot-up time. This overhead is negligible for long-term scheduling, but it can become a problem when running a virtual machine is unnecessary. A scheduling algorithm may terminate a running task to save cost, but it can be restarted to meet the executing task. The overhead caused by launching a new virtual machine may not justify the cost. The time which is determined by BUT(VM_k) is effective at the initial run of

the virtual machine and it can also affect the monetary cost [87].

The value of Earliest Finish Time function, $EFT(t_i, VM_k)$ for each task t_i , whether it is duplicated or an original on the virtual machine VM_k is calculated by adding two values $EST(t_i, VM_k)$ and $ET(t_i, VM_k)$ which Eq. (12) shows.

$$EFT(t_i, VM_k) = EST(t_i, VM_k) + ET(t_i, VM_k). \tag{12}$$

The function $AFT(t_i)$, in Eq. (11), is utilized for the actual finish time of the task t_i which can be measured via Eq. (13) and also Eq. (15).

$$AFT(t_i, VM_{\text{index}}) = \min_{\forall VM_k \in VMs} \{ EFT(t_i, VM_k) \}. \tag{13}$$

The term VM_{index} is used for calculating $AST(t_i, VM_{\text{index}})$ according to Eq. (14). The term VM_{index} indicates task t_i is started on it. The term $AFT(\cdot)$ can be measured via Eq. (15).

$$AST(t_i) = EST(t_i, VM_{\text{index}}), \tag{14}$$

$$AFT(t_i, VM_{\text{index}}) = AST(t_i) + ET(t_i, VM_{\text{index}}). \tag{15}$$

In this paper, the first objective function is to minimize the *makespan* parameter which is determined by Eq. (16).

$$\text{makespan} = \min\{\max_{\forall t_i \in T} (\text{AFT}(t_i))\}. \quad (16)$$

Pricing model

Cloud computing follows the pay-per-use pricing model which means users being charged for the whole time duration even if they use only a fraction of it. Thus, in the proposed model, each instance of leased VM_k is charged per hourly time interval [88–93]. Infrastructure as a service (IaaS) providers offer instances of the virtual machine from the set of available VMs to its clients. Each VM has different configurations like memory size, CPU type, and cost per time unit. The VM configuration determines that the cost of faster VMs are costlier as compared to the slower ones. The function $EUnit(VM_k)$ shows monetary price that must be paid per each unit of execution time on VM_k . Also, the function $EC(t_i, VM_k)$ shows that the fee to be paid for each t_i execution on VM_k according to Eq. (17) similar to literatures in [88, 94, 95]. In Eq. (17), the term $EFT(t_i, VM_k)$ indicates the time interval needed for execution of task t_i on VM_k which is gained via Eq. (12). The upper bound is used to round the execution time because the payment of VMs is based on unit of time in cloud environment. For instance, if one deploys an a1.medium instance from Amazon EC2 which attributed with vector a1.medium = (vCPU = 1, Mem = 2 GiB); its on-demand hourly rate is \$0.0255/h [89]. So, the $EUnit(VM_k)$ value is \$0.0255/hour. If one deploys such on-demand VM for 20 h and 35 min duration, he/she must pay for 21 h. Consequently, the bill is \$0.5355.

$$EC(t_i, VM_k) = \lceil ET(t_i, VM_k) \rceil * EUnit(VM_k). \quad (17)$$

The function $Tunit(VM_k, VM_l)$ is used to show the monetary price which must be paid per each unit of communication data between VM_k and VM_l . Also, the term $TC(t_i, t_j)$ shows the total fee to be paid for communication cost from tasks t_i to t_j between VM_k and VM_l provided each task is assigned on different virtual machines VM_k and VM_l where $k \neq l$, according to Eq. (18).

$$TC(t_i, t_j) = \lceil TT(t_i, t_j) \rceil * Tunit(VM_k, VM_l). \quad (18)$$

Since this article considers hybrid cloud architecture as infrastructure to execute workflows, both private and public cloud must be taken into consideration. Note that, the pricing procedures of the two clouds are different. The private VM scheduler uses an on-demand provisioning approach. The term on-demand is the standard model that is considered by most existing scheduling techniques [88, 96]. Metered services, also called pay-per-use models, are

any type of payment structures in which customers have access to potentially unlimited resources but only pay for what they use. VM instances can be launched and terminated at any time. Because of this, scheduling algorithms need to estimate an optimal number of instances to be allocated before begin of execution provided it is not specified by users; so, it must determine additional instances if it is required during the execution. In this line, those instances that no longer contribute to the workflow execution should also be switched off to save cost. On the other hand, the public VM scheduler uses the charge period provisioning approach. With this assumption, scheduling algorithms try to exploit all the leftover resource utilization of the charged periods and decide whether to terminate machine instances at the end of their charge periods to minimize the cost. With this fine-grained assumption, the scheduler should be aware of fitting tasks within the charged period to take cost-effective procedure; this makes it complicated as considering these points during the scheduling process [96].

Therefore, the amount of total monetary cost (ToC) variable comprises both total execution and transfer costs which are brought in Eq. (19). So, the second objective function is to minimize the ToC cost function.

$$\text{ToC} = \text{TEC} + \text{TTC}. \quad (19)$$

The total execution cost (TEC) variable represents the amount of monetary cost that will be paid for the execution of all tasks in a workload. The TEC value varies depends on one uses the public virtual machine $VM_k^{pu} \in \{\text{public VMs}\}$ or private virtual machine $VM_k^{pr} \in \{\text{private VMs}\}$ because private VM scheduler applies on-demand provisioning and public VM scheduler applies charge period provisioning approaches respectively. According to Eq. (20), the TEC value is measured whether private cloud or public cloud is applied. Thus, for private VM usage, this cost value is calculated in such a way that this is the cost of each task that is executed within $(\forall VM_k^\beta | \beta \in \text{private VM})$ and for public VM cost calculation is for the time between the period start and end time of using $(\forall VM_k^\beta | \beta \in \text{public VM})$. The decision variable x_k is used to indicate whether VM_k is utilized or not.

$$\begin{aligned} \text{TEC} = & \sum_{VM_k \in \text{private VM}} \\ & \sum_{t_i \in VM_k.\text{List}} \lceil EC(t_i, VM_k) \rceil .x_k \\ & + \sum_{VM_k \in \text{public VM}} \lceil (EFT(t_{\text{Last}}, VM_k) \\ & - EST(t_{\text{First}}, VM_k)) * EUnit(VM_k) \rceil .x_k. \end{aligned} \quad (20)$$

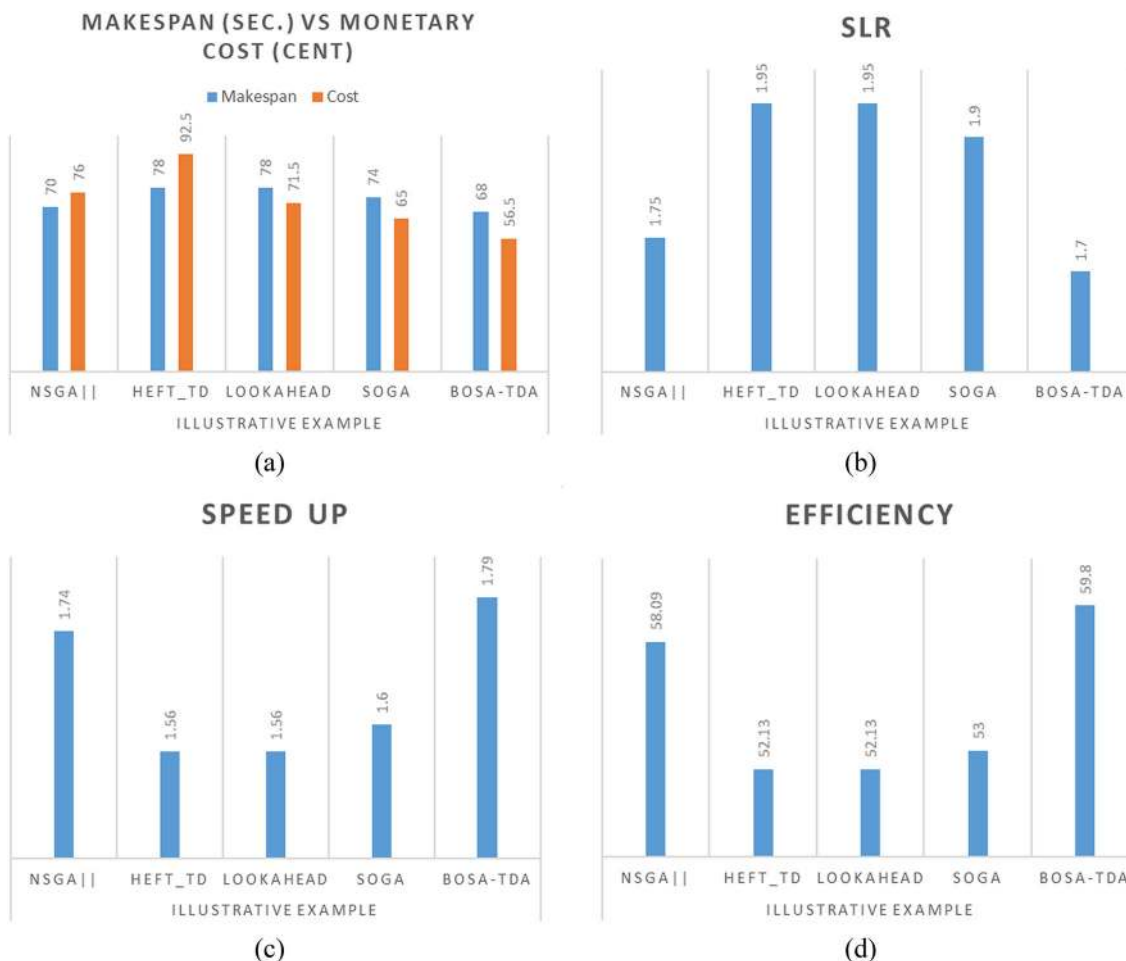


Fig. 7 Comparison of proposed BOSA-TDA against other approaches in terms of evaluation metrics: **a** makespan and monetary, **b** SLR (c), speedup **(d)**, and efficiency for an illustrative example

The terms t_{Last} and t_{First} indicate to the end of the last task execution time and the start of the first task on VM_k . Also, such as TEC, the measurement of TTC variable in hybrid cloud environment has different values depending on which type of underlying public/private virtual machines are applied. Only in the case of communication is inside or starts from public VMs ($VM_l^\beta | \beta \in \text{public VM}$), the transfer monetary cost can be ignored. Equation (21) demonstrates how to calculate the TTC variable.

$$TTC = \sum_{\substack{VM_k \in \text{private VMs}, VM_l \in \text{public VMs} \\ VM_k \neq VM_l}} e(t_i, t_j) \sum_{\substack{t_i \in VM_k\text{-List} \\ t_j \in VM_l\text{-List}}} TC(VM_k, VM_l). \tag{21}$$

Note that, the proposed pricing model is a general model in which it can be utilized for both private cloud owners and individuals without any infrastructure, but both are on a tight budget intend to utilize the public cloud. For the sake of simplicity, in the simulation, only public cloud adoption

is taken into consideration. To this end, the part associated with a private cloud is overlooked from the pricing model, although it does not have any side effect on the proposed model.

An illustrative example

To present the effectiveness of the proposed model, a sample DAG depicted in Fig. 4 is considered. Then, the results relevant to executions of different approaches are reported. This sample graph consists of ten tasks from t_1 to t_{10} . The data transfer time between the tasks is shown by the number above each arc. For executing the tasks in this workflow, a set $VMs = \{VM_1, VM_2, VM_3\}$ of three heterogeneous VMs is considered in this model. The values of functions $\overline{ET}(t_i)$, $AST(t_i^\gamma)$, $AFT(t_i^\gamma)$, $AUV(t_i^\gamma)$, $Ranku(t_i)$, and $Rankd(t_i)$ for each task t_i are given in Table 3. Note that, the lease time interval and boot-up time of the VMs are taken to be zero in this illustrative example. Table 3 also illustrates

the task t_i 's execution time and monetary cost on each VM $_j$.

For encoding the candidate solution, it has two parts; namely, the first part is the task list and the second part is the number of corresponding duplicated tasks. Figure 5 depicts the valid chromosome relevant to a DAG drawn in Fig. 4. In this Figure, tasks t_1 and t_4 were duplicated two times. Note that C_{\max} notation is used for workflow's maximum completion time or *makespan*.

Figure 6 illustrates a comparison between the novel BOSA-TDA algorithm versus several proposed algorithms in the literature.

As Fig. 6 demonstrates BOSA-TDA (depicted in Fig. 6e) outperforms others in terms of *makespan*. After the BOSA-TDA, NSGAI (Fig. 6a), Single objective GA (SOGA) (Fig. 6d), HEFT-TD (Fig. 6b), and Lookahead HEFT-TD (Fig. 6c) have the next ranking order in term of *makespan*. Note that, SOGA only intends to minimize *makespan* metric in which it neglects to take *cost* reduction improvement; then, based on its solution, the cost of utilized VMs is calculated and reported. Usually, it has a good result in the first objective function. In this regards, Fig. 7 depicts the effectiveness of BOSA-TDA in comparison with other approaches in terms of monetary cost, *SLR*, *speedup*, and *efficiency* which are evaluation metrics.

Regarding Fig. 7, it can be concluded that BOSA-TDA outperforms other approaches in terms of evaluation metrics.

Problem statement

The problem, in this paper, is to map a set of tasks of a given workload on a set of VM instances in such a way that total monetary cost (ToC) and *makespan* of the workflow scheduling are simultaneously minimized while some constraints are preserved. This problem is formulated to a bi-objective optimization problem with service cost and service time minimization viewpoints; this formulation is drawn in Eqs. (22–26). Note that the first and the second objectives have been elaborated in Eq. (16) and Eq. (19) respectively.

$$\text{minimize } F(\overrightarrow{\text{solution}}) = [f_{\text{Makespan}}(\overrightarrow{\text{solution}}), f_{\text{ToC}}(\overrightarrow{\text{solution}})], \quad (22)$$

$$f_{\text{Makespan}}(\overrightarrow{\text{solution}}) : \min(\{\max_{\forall t_i \in T} (\text{AFT}(t_i))\}), \quad (23)$$

$$f_{\text{ToC}}(\overrightarrow{\text{solution}}) : \min(\text{TEC}(\overrightarrow{\text{solution}}) + \text{TTC}(\overrightarrow{\text{solution}})), \quad (24)$$

$$s.t : \text{ToC} \leq \text{Budget}, \quad (25)$$

$$\text{makespan} \leq \text{Deadline}. \quad (26)$$

In this formal presentation, two constraints (25, 26) can be adjusted by users depending on their monetary budget and time sensitivity. Since the objectives in a bi-objective optimization problem usually conflict with each other, the Pareto dominance concept is commonly used to compare generated solutions [87].

Proposed bi-objective optimization based on simulated annealing task duplication scheduling algorithm (BOSA-TDA)

The problem statement in the previous section reveals some important points. Namely, the issue we face is an NP-Hard problem with discrete search space. In addition, it is a multi-objective optimization problem the reason why it needs profound exploring search space to find abundant Pareto solutions. To have abundant solutions, the SA suffers to provide a handful of candidate solutions since the canonical SA is a point-wise algorithm. So, a new population-based BOSA-TDA algorithm is devised that takes benefit of HEFT approaches, i.e., upward, downward, and level rankings, and also duplicated list of tasks in their initial population. Algorithm 1 illustrates the novel proposed BOSA-TDA.

Algorithm 1: Bi-objective Simulated Annealing Task Duplication Scheduling Algorithm (BOSA-TDA)**Input:** A DAG and underlying platform specifications; $nPop$ as population size, and $Freeze$ as SA parameter**Output:** Set of non-dominated scheduling solutions

```

Begin
1.  $InitialTemp = 1000$ 
2.  $Freeze = 0$ 
3.  $a = 0.5$ 
4.  $Pop [1..nPop]$  : Population of  $nPop$  individuals
5.  $NewPop [1..nPop]$  : New Population of  $nPop$  individuals
6.  $PS = Null$ ; /* PS is Pareto Set of non-dominated solutions
7. Create new  $Pop[1]$ .Chrom according to  $Ranku(.)$  with ascending order
8. Create new  $Pop[2]$ .Chrom according to  $Rankd(.)$  with descending order
9. Create new  $Pop[3]$ .Chrom according to  $Level(.)$  with descending order
10. Calculate  $Pop[1].Obj1$ ,  $Pop[2].Obj1$ , and  $Pop[3].Obj1$  via Eq.(16)
11. Calculate  $Pop[1].Obj2$ ,  $Pop[2].Obj2$ , and  $Pop[3].Obj2$  via Eq. (19)
12. For  $i = 4$  To  $nPop$ 
13.   Create  $Pop[i]$ .Chrom according to CreateRandomSolution in Algorithm 2
14.   Calculate  $Pop[i].Obj1$  via Eq. (16)
15.   Calculate  $Pop[i].Obj2$  via Eq. (19)
16. End for
17. For  $IT = 1$  To  $MaxIteration$ 
18.    $MaxObj1 = Max(Pop.obj1)$  // For Each Object find Max Object in population
19.    $MinObj1 = Min(Pop.obj1)$ 
20.    $MaxObj2 = Max(Pop.obj2)$ 
21.    $MinObj2 = Min(Pop.obj2)$ 
22.   For  $i = 1$  To  $nPop$ 
23.      $temp = InitialTemp$  /* Initialize temperature
24.     While ( $temp > Freeze$ )
25.       /*  $newPop[i]$ .Chrom = Create new chromosome using Create neighbor( $Pop[i]$ .Chrom)
26.       /* Start Neighbor Search
27.       Draw an integer value  $R$  in  $[1..4]$ ;
28.       if  $R=1$  then  $newPop[i]$ .Chrom is created by Calling Algorithm 3 (CrossoverTask on  $Pop[i]$  and
29.          $Pop[rand]$ )
30.       else if  $R=2$  then  $newPop[i]$ .Chrom is created by Calling Algorithm 4 (CrossoverDup. on  $Pop[i]$  and
31.          $Pop[rand]$ )
32.       else if  $R=3$  then  $newPop[i]$ .Chrom is created by Calling Algorithm 5 (MutationTask on  $Pop[i]$ )
33.       else  $newPop[i]$ .Chrom is created by Calling Algorithm 6 (MutationDup. On  $Pop[i]$ )
34.       /* End of Neighbor Search
35.        $newPop[i].Obj1$  is calculated via Eq. (16)
36.        $newPop[i].Obj2$  is calculated via Eq. (19)
37.        $NormalNewPop[i].obj1 = \left( \frac{NewPop[i].obj1 - MinObj1}{MaxObj1 - MinObj1} \right)$ 
38.        $NormalNewPop[i].obj2 = \left( \frac{NewPop[i].obj2 - MinObj2}{MaxObj2 - MinObj2} \right)$ 
39.        $NormalPop[i].obj1 = \left( \frac{Pop[i].obj1 - MinObj1}{MaxObj1 - MinObj1} \right)$ 
40.        $NormalPop[i].obj2 = \left( \frac{Pop[i].obj2 - MinObj2}{MaxObj2 - MinObj2} \right)$ 
41.        $UtilizedNewPop = (a * NormalNewPop[i].obj1) + ((1 - a) * NormalNewPop[i].obj2)$ 
42.        $UtilizedPop = (a * NormalPop[i].obj1) + ((1 - a) * NormalPop[i].obj2)$ 
43.        $\Delta newPop = (UtilizedNewPop[i] - UtilizedPop[i]) / UtilizedPop[i]$ 
44.       If  $dominate(newPop[i], Pop[i]) = True$  /* Select better solution
45.          $Pop[i] = newPop[i]$ 
46.       Else
47.         If  $rand < \exp(-\Delta newPop / temp)$ 
48.            $Pop[i] = newPop[i]$  /* Select worse solution with probability
49.         end if
50.       end if
51.        $temp = 0.99 * temp$  /* cooling phase
52.     End while
53.      $Temp = PS \cup Pop \cup newPop$  // Add New Population To main Population
54.   End for
55.    $[PS, F] = Nondominated\ sorting(Temp)$ 
56.   /* calculate crowding distance and select based on ref. [81] */
57.    $PS = CrowdingDistance(PS, F)$ 
58. End for
59. Return  $PS$ 
60. END

```

This pseudo-code in Algorithm 1 is relevant to the main algorithm; it receives a DAG and underlying hybrid platform specifications; then, it returns a set of non-dominated solutions as PS . Since diverse solutions are preferable against dense solutions in multi-objective problems, before the main algorithm returns the final non-dominated solutions which were gained from the first front-ranking, it calls the *CrowdingDistance* procedure to find diverse solutions such as in [81]. Similar to other population-based algorithms, the proposed algorithm starts with random initial individuals. Note that each individual in the population is a record of three fields. Namely, the fields *Chrom*, *Obj1*, and *Obj2* which are used for solution encoding, the first and the second objectives respectively. In this line, the first three individuals of the population are created according to upward, downward, and level ranking approaches, for the rest of the population the *CreateRandomSolution* algorithm is called to produce other individuals, which is depicted in Algorithm 2. After the individuals are prepared, the objective functions can be calculated via Eq. (16) and Eq. (19) respectively. Then, it plunges into the main loop of Algorithm 1 through lines (17–56). This repeats *MaxIteration* times which is set 50 times in this paper. In each iteration, for all individuals of the population, several instructions are run. For each individual as a candidate, the SA is run to explore the neighborhood of each solution. To do so, a neighborhood operation is defined which calls randomly one of the four algorithms: Algorithm 3 through Algorithm 6; by this, search space will be efficiently permuted. If a new solution is better than the current one, it is accepted. In the other words, if the new solution dominates the previous one it is substituted in line#43, otherwise the worse solution is probabilistically accepted in line #46. The SA utilizes the temperature concept, *temp*, to take over the algorithm. This is set the *temp* to the big value at the outset and it gradually decreases it to reach freeze value. The exponential function $\exp(-\Delta newPop/temp)$ is used to calculate the probability of worse solution acceptance; the temperature *temp* value is near to freeze, the chance of acceptance is near to 0. The parameter $\Delta newPop$ is used to have the effectiveness of normalized objective functions in the decision. The coefficient α is applied to indicate the importance of objective functions. For now, this article takes 0.5 as the same importance for each of both. After that, the small cooling stage happens to reach in stable point. Since the range of two objective functions is different, the individuals' objective functions are normalized based on the minimum and maximum of both objective function values of the whole population in each round.

Initial population

The proposed BOSA-TDA similar to other meta-heuristic computations starts with the initial population. As the main algorithm depicts, the first three individuals of the population are placed by upward, downward, and level ranking algorithms which are HEFT-based approaches. For the rest individuals, Algorithm 2 is called which has random treatment. It guarantees the casual traversing of search space. For this reason, Algorithm 2 is called to create populations with more plenitude solutions. To do so, it generates new chromosomes in two steps. The first step is to create a random order of tasks for the first part of chromosome $\langle TaskList \rangle$ and the second is to create a random number of duplications in $\langle DuplicationList \rangle$ for each task t_r . The initial value of $\langle TaskList \rangle$ is set to null. In this pseudo-code, the list of tasks that have no predecessor, namely $Pred(t_i) = \emptyset$; or input nodes are put in set $\{AvailSet\}$. The variable list $\{AvailSet\}$ is a set of tasks that can be selected at any time for inserting to the list $\langle TaskList \rangle$. In the *CreateRandomSolution* algorithm in lines (4–14), while loop is dedicated to continuing until the set $\{AvailSet\}$ has any member of predecessors. The variable *Visited* is an array that holds the visited task number at any time. In this regard, in line 5, a random task t_r is selected from the set $\{AvailSet\}$. If all predecessors of t_r are visited before then t_r is removed from the set $\{AvailSet\}$. Then, the values $Succ(t_r)$ are added to the set $\{AvailSet\}$. Afterward, the selected task t_r is added to $\langle TaskList \rangle$ otherwise the task t_r is removed from the set $\{AvailList\}$ because t_r in the forthcoming rounds will be added.

After adding all tasks of set T to the list $\langle TaskList \rangle$ separately, it is turned to the second step which is placed in lines (15–19). In this section, there should be a random number assigned to the task that represents the number of duplicates for each task. However, the values assigned to the task list must have at least one and a maximum value of *MaxD* because the value more than this number will only cause additional duplication and burdens redundant monetary costs without any performance improvement.

Algorithm 2: *CreateRandomSolution Procedure***Input:** A DAG specification, nT : Number of tasks**Output:** A chromosome contains tasks in *TaskList* and their duplications in a *DuplicationList***Begin**

```

1. TaskList = []
2. Visited = zeros[1..nT]
3. AvailSet = {All  $t_i$  that  $Pred(t_i) = \emptyset$ }
4. while AvailSet  $\neq \emptyset$ 
5.    $t_r$  = select Random task from AvailSet
6.   if Visited( $\forall$  all  $Pred(t_r)$ ) == 1
7.     AvailSet = AvailSet -  $t_r$ 
8.     AvailSet = AvailSet + Succ( $t_r$ )
9.     TaskList = [TaskList,  $t_r$ ]
10.    Visited[ $t_r$ ] == 1
11.  else
12.    AvailSet = AvailSet -  $t_r$ 
13.  end if
14. End while
15. DuplicationList = []
16. For  $i = 1$  To  $nT$ 
17.    $MaxD = \min(\text{count}\{VMs\}, \max(\text{count}(Pred\{t_i\}), \text{count}(Succ\{t_i\})))$ 
18.   DuplicationList[ $i$ ] = random(1.. $MaxD$ )
19. End for
20. Return TaskList and DuplicationList
21. End

```

Simulated annealing process

As the main body of the algorithm calls an SA-like procedure, it needs some operators to permute discrete search space efficiently. For this, four different neighbors of current state algorithms are applied; each of which is randomly called in every round. As a candidate solution has chromosome encoding which has two parts task list and duplication list, our new operators target both parts. Note that, both task lists and duplication lists are incorporated in proposed meta-heuristic operations. The name of four neighbor algorithms are *CrossoverTask*, *CrossoverDuplication*, *MutationTask*, and *MutationDuplication* Algorithms.

CrossoverTask algorithm

The *CrossoverTask* procedure receives two chromosomes from the Population in the form of $\langle TaskList \rangle$ as input and crossover them to generate a new chromosome. The first one is $Pop[i]$ and the second one is a random $Pop[rand]$; then, it returns a better child. For doing this, $x1$ and $x2$ are considered as inputs and $Y1$ and $Y2$ are new children. Firstly, the algorithm generates a random number R and acts as a single point crossover of the genetic task scheduling algorithm such as in [8], i.e., it copies $x1[1..R]$ to corresponding elements of $Y1$ and copies $x2[1..R]$ to $Y2$. Then, all tasks in $x2$ that do not exist in $Y1$ will be inserted to $Y1$ in the same order. Also, all tasks in $x1$ that do not exist in $Y2$ will be inserted to $Y2$ in the same order; this procedure guarantees dependency constraints. Then, it returns dominated child. Algorithm 3 is presented to show this type of crossover. Also, Fig. 8 illustrates its functionality.

Algorithm 3: CrossoverTask Procedure

Input: chTx1: task list of the chromosome, and
chTx2: task list of chromosome

Output:

Y: a dominated child with a task list of chromosome

Begin

1. *chTx1 and chTx2 are list of valid execution orders of t_i tasks*
2. $R = \text{random}(1:nT - 1)$
3. $x1 = \text{chTx1}, x2 = \text{chTx2}$
4. $Y1 = x1[1..R], Y2 = x2[1..R]$
5. *For $i = 1$ To nT*
6. *Check if $x2[i]$ not exist in $Y1$ then insert $x2[i]$ to $Y1$*
7. *Check if $x1[i]$ not exist in $Y2$ then insert $x1[i]$ to $Y2$*
8. *End for*
9. *If dominate ($Y1, Y2$) == True then*
10. $Y=Y1;$
11. *Else*
12. $Y=Y2;$
13. *End if*
14. *Return Y as the better child*

END

CrossoverDuplication algorithm

The CrossoverDuplication procedure receives two chromosomes from the population in the form of $\langle \text{DuplicationList} \rangle$ as input and crossover them to generate a new chromosome. The first one is Pop[i] and the second one is a random Pop[random]; then, it returns a better child. For doing this, firstly it generates a random number R; then, it puts values [chD1[1..R], chD2[R + 1..nT]] in Y1, and also it puts values [chD2[1..R], chD1[R + 1..nT]] in Y2. Algorithm 4 depicts the *CrossoverDuplication* procedure. Also, Fig. 9 illustrates its functionality.

Algorithm 4: CrossoverDuplication Procedure

Input: chD1: Duplication list of the chromosome, and
chD2: Duplication list of chromosome

Output:

Y: the dominated child with Duplication list

Begin

1. *chD1 and chD2 are list of valid Duplications of t_i tasks*
2. $R = \text{random}(1:nT - 1)$
3. $Y1 = [\text{chD1}[1..R], \text{chD2}[R + 1..nT]]$
4. $Y2 = [\text{chD2}[1..R], \text{chD1}[R + 1..nT]]$
5. *If dominate ($Y1, Y2$) == True then*
6. $Y=Y1;$
7. *Else*
8. $Y=Y2;$
9. *End if*
10. *Return Y as the better child*

END

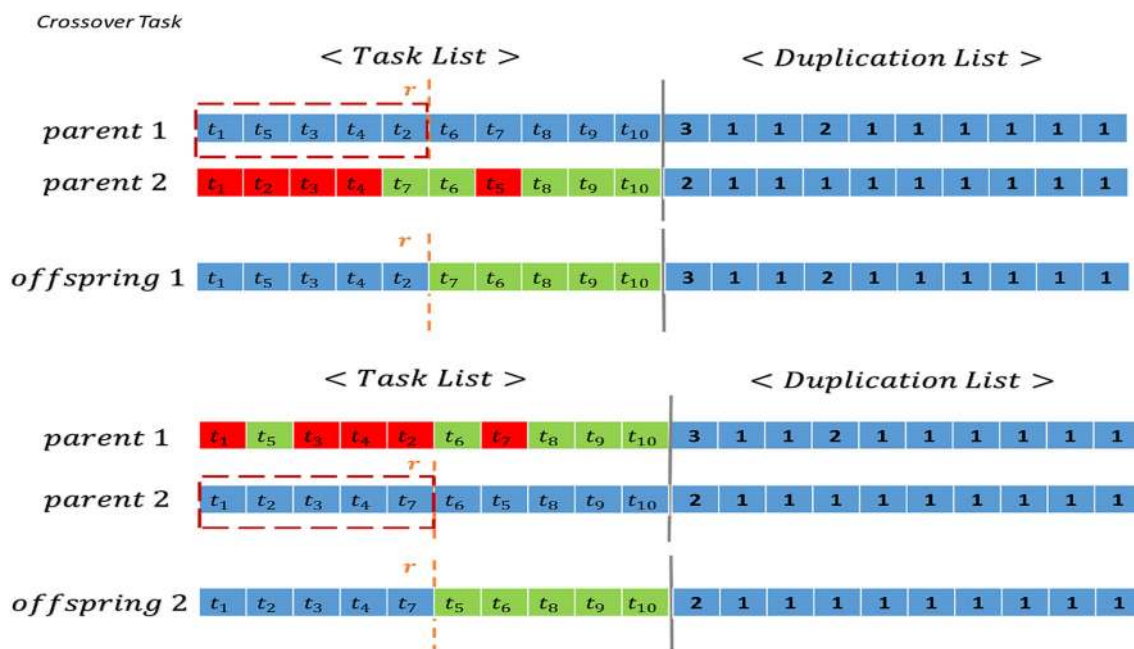


Fig. 8 Example of crossover task procedure

MutationTask procedure

This procedure receives a chromosome chT from $\langle TaskList \rangle$ as input and mutates it to generate a new chromosome to $\langle TaskList \rangle$. For doing so, this firstly generates random number i ; then, it checks all elements after place $chT[i]$ until it reaches the end of the list or reach a member t_j that is the successor of $chT[i]$. Then,

in line #10 of Algorithm 5, it selects a random number h between $chT[i + 1..j - 1]$, if all predecessor of $chT[h]$ is in $chT[1..i - 1]$ (call it *HeadTasks*), it swaps $chT[h]$ and $chT[i]$ values. Algorithm 5 depicts the MutationTask procedure. Also, Fig. 10 illustrates its functionality.

Algorithm 5: MutationTask procedure

Input: A chromosome include and chT : task list of chromosomes

Output:
IsMutated: Boolean value
Mutated_chT: task list of chromosomes

Begin

1. $IsMutated = False$
2. $i = random(1..nT)$
3. **For** $j = i + 1$ **To** nT
4. **If** $t_j \in succ\{t_i\}$
5. break
6. **End if**
7. **End for**
8. **If** $(j - i > 1)$
9. $HeadTasks = chT[1..i - 1]$
10. $h = random(i + 1..j - 1)$
11. **if** $((pred\{t_{chT(h)}\} \cap HeadTasks) = pred\{t_{chT(h)}\})$ // all predecessor of $chT[h]$ is in *HeadTasks*
12. $Muted_chT = [chT[1..i - 1], chT[h], chT[i + 1..h - 1], chT(i), chT[h + 1..nT]]$
13. $IsMutated = true$
14. **End if**
15. **End if**
16. Return *Mutated_chT*

END

MutationDuplication procedure

Such as all evolutionary algorithms the operation of mutation may help to avoid getting stuck in local optimal; this is the reason to customize mutation operation in both task and duplication lists. By getting a chromosome, the value of its $\langle DuplicationList \rangle$ part is mutated. This function randomly mutates one or more members of the list $\langle DuplicateList \rangle$. Also, the new values do not violate the minimum and maximum conditions. Algorithm 6 depicts the *MutationDuplication* procedure. In addition to, Fig. 11 illustrates its functionality.

Algorithm 6: MutationDuplication Procedure

Input: A chromosome and
chD : A duplication list of chromosomes

Output:
DuplicationList : duplication list of chromosomes

Begin

1. *DuplicationList* = *chD*
2. $\{TSet\} = \text{randsample}(1: nT, 2)$
3. For each $t_i \in \{TSet\}$
4. *DuplicationList*(t_i) = randomly increase or decrease (t_i)
5. $MaxD = \min(\text{count}\{VMs\}, \max(\text{count}\{Pred\{t_i\}\}, \text{count}\{Succ\{t_i\}\}))$
6. correct *DuplicationList*(t_i) according to minimum 1 and maximum *MaxD*
7. End for
8. Return *DuplicationList*

END

virtual machines. Unfortunately, network delay has a drastic impact on execution time; this is the reason the scheduler must be aware of the workload nature. The edges of the DAG representing the dependencies can be weighted to indicate the data transfer requirements. In this regard, the communication to computation ratio (CCR) of a graph is used for knowing how to extend the workload communication-intensive or computation-intensive is and to define how long the data transfer on the network will take. This CCR parameter is calculated via Eq. (27). It is the ratio of the average communication cost to the average computation cost. If a DAG's CCR value is very low, it can be considered as a computation-intensive application [1, 19].

Simulation and evaluation

This section is dedicated to the simulation and evaluation of the novel BOSA-TDA. To do so, miscellaneous scenarios are conducted to reach robust results. As such, different scenarios and datasets are generated. Forthcoming subsections are considered for scheduling metrics; scenarios; and datasets; experiments; and data analysis for this reason. Each scenario has been independently executed 20 times each of which sets *MaxIteration* = 50 in its main loop; then, the average of execution results was reported.

Scheduling metrics

To have better evaluation and comparisons, several metrics are applied which are being pervasively used in literature. These metrics are enlisted in this section.

Communication to computation ratio (CCR)

The unpleasant feature of network delay is considered in this model whenever there needs to be communication between

$$\text{CCR} = \frac{\frac{1}{|E|} \sum_{\forall e(t_i, t_j) \in \{E\}} \overline{e(t_i, t_j)}}{\frac{1}{nT} \sum_{\forall t_i \in \{T\}} \overline{ET(t_i)}} \quad (27)$$

Schedule length ratio (SLR)

Since each graph has its feature to be utilized in the scheduling process, the *makespan* gained as a result of each algorithm is no longer meaningful. To obviate this problem, considering a lower bound for execution time is a beneficial approach to give a bright clue. To avoid confusion, it needs to be normalized. To do so, it leverages the critical path (*CP*) concept to make a new parameter schedule length ratio (*SLR*). The graph's *CP* is the longest path from *entry* node to *exit* node in which it cannot be parallelizable owing to dependency. If the scheduler executes nodes relevant to *CP* on the fastest available processors/VMs, the *makespan* parameter cannot be less than *CP*'s length. Then, the new parameter *SLR* which is a normalized metric regardless of the studied graph can be measured via Eq. (28). This metric is defined relative to the critical path rather than the total execution time. This is because the shortest execution time of a job on a highly parallel platform is determined by the length of its critical path. Note that, *SLR* value of each schedule is greater than one. If

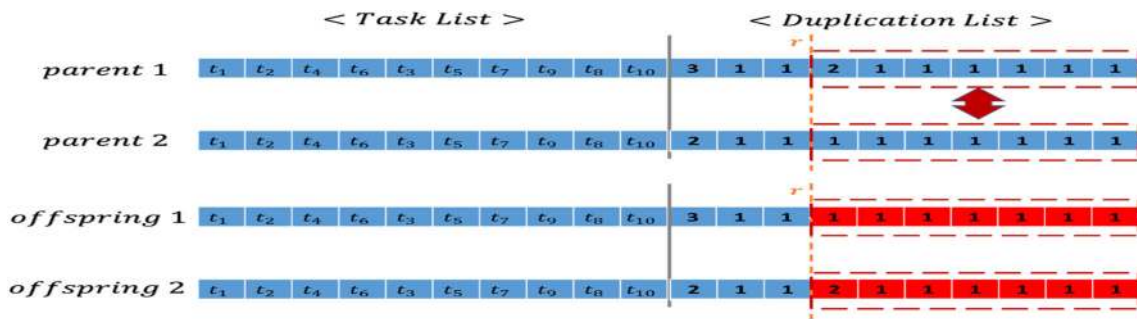


Fig. 9 Example of crossover duplication procedure

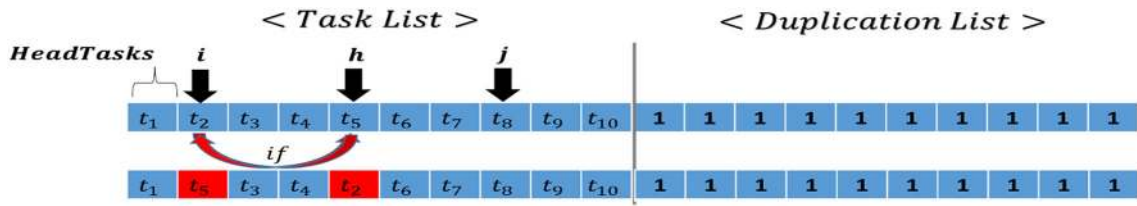


Fig. 10 Example of mutatetask procedure

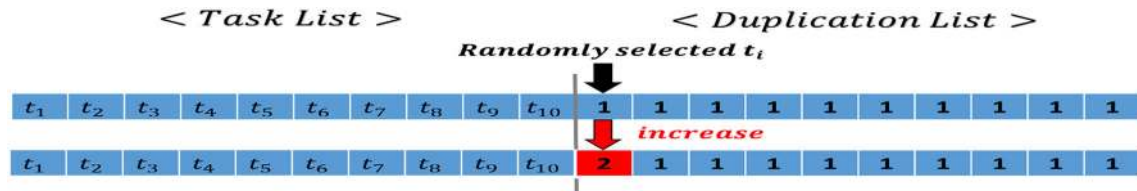


Fig. 11 Example of mutate duplication procedure

this value is close to one, then it can be said that the scheduler is working very well.

$$SLR = \frac{Makespan}{\sum_{t_i \in \text{Critical Path}} \min(ET(t_i, VM_k)) \forall VM_k \in \{VMs\}} \quad (28)$$

Speed up

This metric indicates how many times the algorithm runs faster in comparison to on a single processor or a VM, preferably the faster processor [1]. This metric is attained via Eq. (29).

$$\begin{aligned} \text{Speed up} &= \frac{\text{serial execution on the fastest VM}}{Makespan} \\ &= \frac{\min_{\forall VM_k \in \{VMs\}} \left(\sum_{t_i \in T} ET(t_i, VM_k) \right)}{Makespan} \quad (29) \end{aligned}$$

Efficiency

The complementary metric is *efficiency* because the *speed up* metric does not determine you gained this level of speed up with spending how many processors [1]. This metric is calculated via Eq. (30).

$$\text{Efficiency} = \frac{\text{speed up}}{\text{number of used VM}} * 100\%. \quad (30)$$

Scenarios and datasets

To assess the performance of the proposed BOSA-TDA, the famous workloads such as synthetic data from five real-world scientific workflow applications are used [47], such as Montage (generation of image mosaics of the sky), Epigenomics (mapping of the epigenetic state of human cells), CyberShake (generating seismic hazard maps for earthquake detection), LIGO (detection of gravitational waves in the universe), and SIPHT (used in biology) [5]. In this line, the evaluation is based on the comparison between the proposed algorithm against other state-of-the-arts in terms of prominent evaluation metrics of scheduling domain such as *makespan*, *cost*, *SLR*, *speedup*, and *efficiency*. Moreover, to demonstrate the

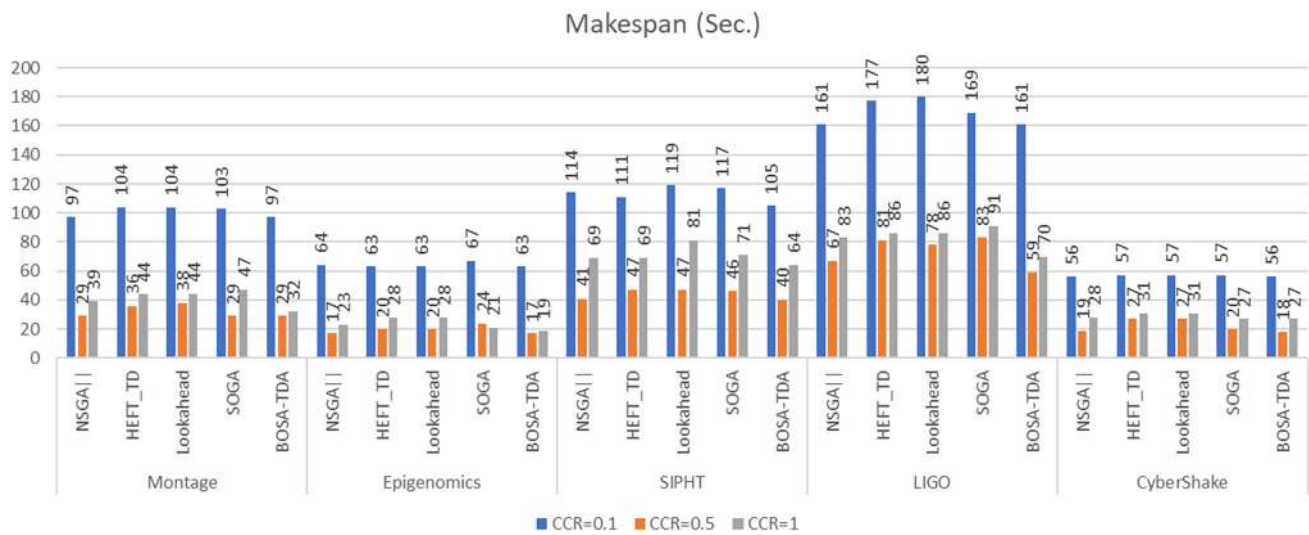


Fig. 12 Comparison of all comparative algorithms in term of *makespan* value in all scenarios for different workloads

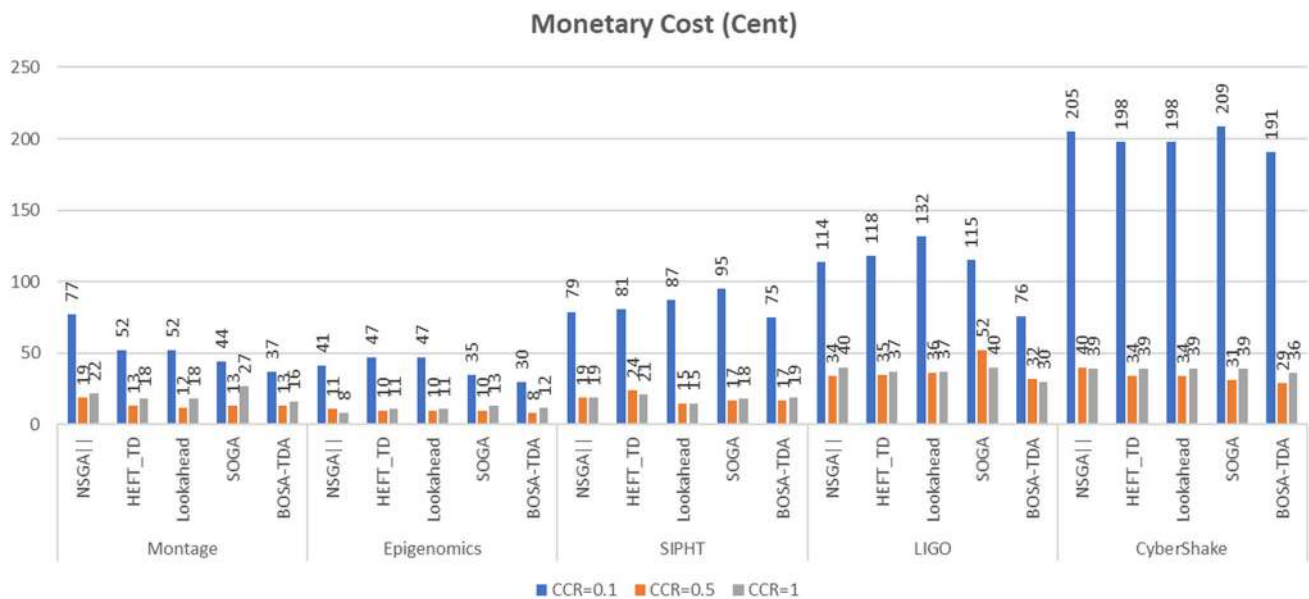


Fig. 13 Comparison of all comparative algorithms in term of monetary cost value in all scenarios for different workloads

effectiveness of novel BOSA-TDA on a wide workload spectrum with a different attribute from computation-intensive to communication-intensive graphs, a new extension of such graphs is produced. Since different VMs' configurations have variable performance and pricing schemes, different correlations between variables VM speed and VM price are utilized. The VMs' processing power is considered such as Amazon EC2 instances A, T, M, etc. [93]. In addition to, the pricing scheme is considered hourly range from \$0.011/h to \$0.27/h (\$94/year to \$2367/year) [92]. Since some clouds neglect in/out data transfer charges and for the sake of simplicity, the data transfer cost and model constraints are omitted. In this dataset, it is taken a positive correlation coefficient near to 1

value for *ES* and *EC* vectors. This indicates that once execution speed increases, the execution cost also increases. Also, the datasets are conducted in such a way that having diverse workloads in terms of *CCR*; in this way, the robust evaluation can be made. To do so, four different values for *CCR* which are 0.1, 0.5, 1.0, and 5.0 considered respectively for computation-intensive, rather computation-intensive, moderate, and communication-intensive graphs. As the result of extensive simulations for communication-intensive graphs with parameter *CCR* equal and more than 5 proves that there is not any meaningful discrepancy between the performance of comparative algorithms, the last condition is ignored. Note that, in case of last condition, all algorithms behave such as a

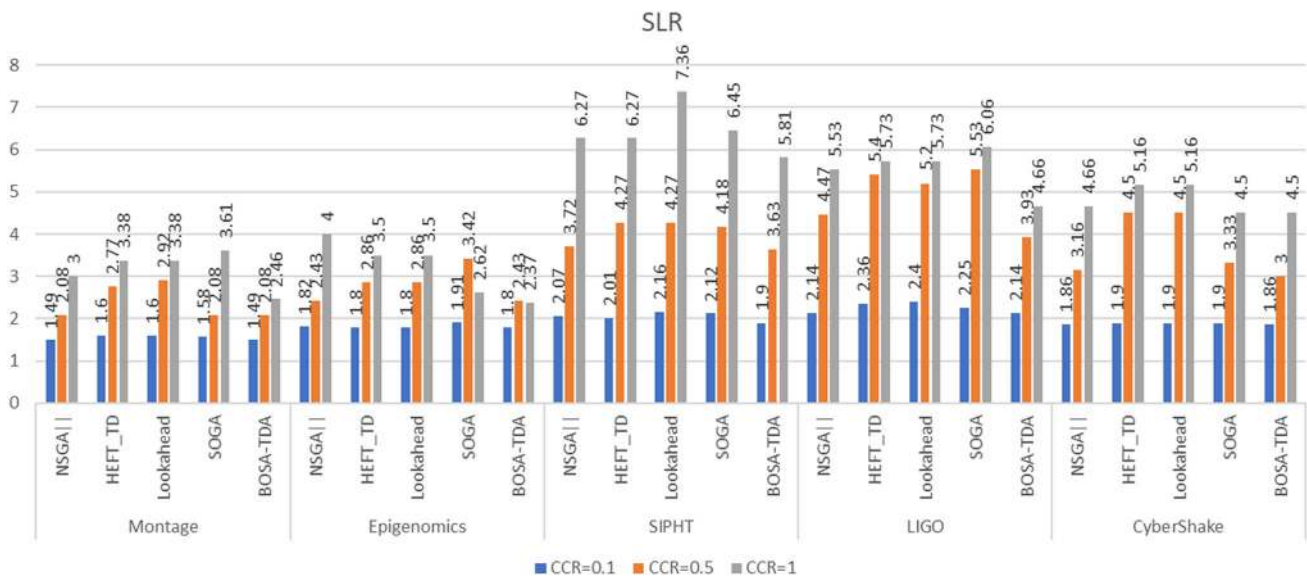


Fig. 14 SLR comparison of all comparative algorithms in all scenarios

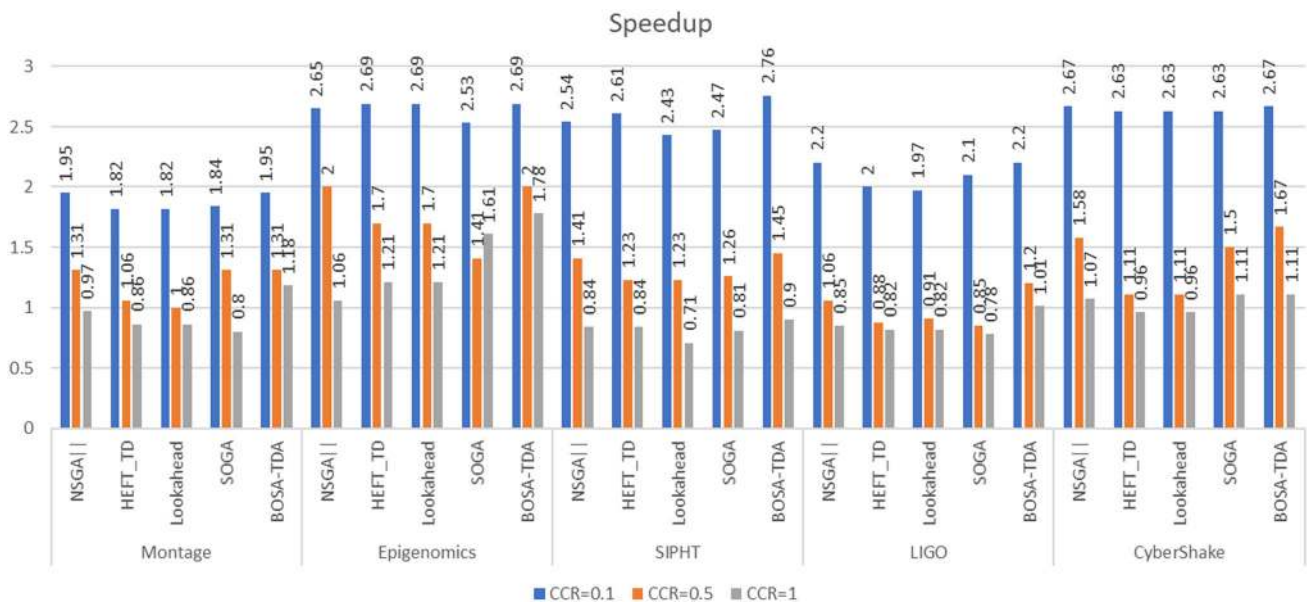


Fig. 15 The speedup comparison of all comparative algorithms in all scenarios

serial algorithm [1]. Overall, 15 scenarios have been considered to be examined for the efficiency of the current proposed work. Namely, there are five famous workloads each of which with three different CCR parameters.

Experiments and data analysis

Experiments

In this section, experimental results that are drawn from extensive simulations of proposed BOSA-TDA in comparison with state-of-the-arts NSGAII [81], HEFT-TD [30],

Lookahead [30], and SOGA [85] are reported. Note that, the average result of 20 independent executions is reported in terms of *makespan*, total monetary costs, *SLR*, *speed up*, and *efficiency* metrics. Figures 12 through 16 are dedicated to these contrasts. As such, Figs. 12 and 13 respectively depict algorithms comparison in terms of *makespan* and monetary costs separately relevant to computation-intensive (CCR = 0.1), rather computation-intensive (CCR = 0.5), and moderate (CCR = 1.0) of understudy workloads. Figure 12 is condensed to show the comprehensive behavior of comparative algorithms in term of *makespan* value of all scenarios.

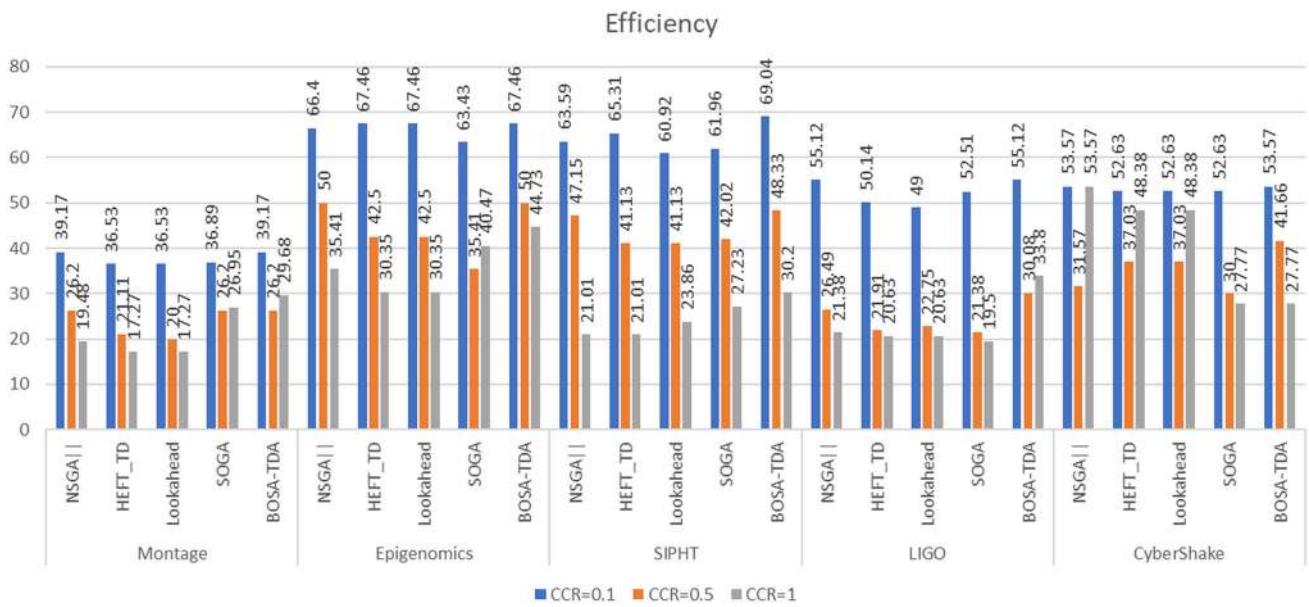


Fig. 16 The efficiency comparison of all comparative algorithms in all scenarios

Table 4 Then comparison of BOSA-TDA versus state-of-the-arts in term of RPD of makespan

Dataset	CCR	Makespan (s)					Relative percent deviation (RPD)			
		NSGAII	HEFT_TD	Lookahead	SOGA	BOSA-TDA	NSGAII (%)	HEFT_TD (%)	Lookahead (%)	SOGA (%)
Montage	0.1	97	104	104	103	97	0	7	7	6
Montage	0.5	29	36	38	29	29	0	19	24	0
Montage	1.0	39	44	44	47	32	18	27	27	32
Epigenetics	0.1	64	63	63	67	63	2	0	0	6
Epigenetics	0.5	17	20	20	24	17	0	15	15	29
Epigenetics	1.0	23	28	28	21	19	17	32	32	10
SIPHT	0.1	114	111	119	117	105	8	5	12	10
SIPHT	0.5	41	47	47	46	40	2	15	15	13
SIPHT	1.0	69	69	81	71	64	7	7	21	10
LIGO	0.1	161	177	180	169	161	0	9	11	5
LIGO	0.5	67	81	78	83	59	12	27	24	29
LIGO	1.0	83	86	86	91	70	16	19	19	23
CyberShake	0.1	56	57	57	57	56	0	2	2	2
CyberShake	0.5	19	27	27	20	18	5	33	33	10
CyberShake	1.0	28	31	31	27	27	4	13	13	0
Average							6	15	17	12

In this regard, the Fig. 13 is also condensed to illustrate the comprehensive behavior of comparative algorithms in term of monetary cost value of all scenarios.

For computation-intensive workload case studies which have CCR = 0.1, Fig. 12 demonstrates that BOSA-TDA outperforms against other algorithms in term of makespan metric except for in workloads LIGO and Montage that the BOSA-TDA has the same behavior with NSGAII; also, in Epigenomics dataset in which it has the same behavior with

HEFT-TD and Lookahead algorithms; in other cases, the BOSA-TDA has significant dominance against counterparts. For monetary cost analysis, Fig. 13 proves the dominance of BOSA-TDA against other state-of-the-arts in all cases.

In addition to, for rather computation-intensive graphs which have CCR = 0.5, Fig. 12 proves that BOSA-TDA beats other algorithms in terms of makespan except for in Montage and Epigenomics workloads so that the BOSA-TDA has the same behavior with NSGA II. Also, the BOSA-TDA does

Table 5 The comparison of BOSA-TDA versus state-of-the-arts in term of RPD of monetary cost

Dataset	CCR	Monetary cost (Cent)					Relative percent deviation (RPD)			
		NSGAll	HEFT_TD	Lookahead	SOGA	BOSA-TDA	NSGAll (%)	HEFT_TD (%)	Lookahead (%)	SOGA (%)
Montage	0.1	77	52	52	44	37	52	29	29	15
Montage	0.5	19	13	12	13	13	32	3	− 5	3
Montage	1.0	22	18	18	27	16	30	16	16	43
Epigenetics	0.1	41	47	47	35	30	26	37	37	15
Epigenetics	0.5	11	10	10	10	8	27	22	22	16
Epigenetics	1.0	8	11	11	13	12	− 46	− 8	− 7	8
SIPHT	0.1	79	81	87	95	75	5	7	13	21
SIPHT	0.5	19	24	15	17	17	10	26	− 14	− 2
SIPHT	1.0	19	21	15	18	19	− 1	8	− 26	− 4
LIGO	0.1	114	118	132	115	76	33	36	42	34
LIGO	0.5	34	35	36	52	32	8	11	11	39
LIGO	1.0	40	37	37	40	30	26	20	20	27
CyberShake	0.1	205	198	198	209	191	7	4	4	9
CyberShake	0.5	40	34	34	31	29	27	14	14	7
CyberShake	1.0	39	39	39	39	36	8	7	7	8
Average							16	15	11	16

Table 6 The comparison of BOSA-TDA versus state-of-the-arts in term of RPD of SLR

Dataset	CCR	SLR					Relative percent deviation (RPD)			
		NSGAll	HEFT_TD	Lookahead	SOGA	BOSA-TDA	NSGAll (%)	HEFT_TD (%)	Lookahead (%)	SOGA (%)
Montage	0.1	1.49	1.60	1.60	1.58	1.49	0	7	7	6
Montage	0.5	2.08	2.77	2.92	2.08	2.08	0	25	29	0
Montage	1.0	3.00	3.38	3.38	3.61	2.46	18	27	27	32
Epigenetics	0.1	1.82	1.80	1.80	1.91	1.80	1	0	0	6
Epigenetics	0.5	2.43	2.86	2.86	3.42	2.43	0	15	15	29
Epigenetics	1.0	4.00	3.50	3.50	2.62	2.37	41	32	32	10
SIPHT	0.1	2.07	2.01	2.16	2.12	1.90	8	5	12	10
SIPHT	0.5	3.72	4.27	4.27	4.18	3.63	2	15	15	13
SIPHT	1.0	6.27	6.27	7.36	6.45	5.81	7	7	21	10
LIGO	0.1	2.14	2.36	2.40	2.25	2.14	0	9	11	5
LIGO	0.5	4.47	5.40	5.20	5.53	3.93	12	27	24	29
LIGO	1.0	5.53	5.73	5.73	6.06	4.66	16	19	19	23
CyberShake	0.1	1.86	1.90	1.90	1.90	1.86	0	2	2	2
CyberShake	0.5	3.16	4.50	4.50	3.33	3.00	5	33	33	10
CyberShake	1.0	4.66	5.16	5.16	4.50	4.50	3	13	13	0
Average							8	16	17	12

not have any dominance against SOGA in term of makespan improvement in only Montage workload. In term of monetary cost, as Fig. 13 shows, the BOSA-TDA has improvement in cost reduction against others in the majority workloads except for in Montage and SIPHT workloads where the Lookahead algorithm has dominance against BOSA-TDA. In the aforementioned workloads, only SOGA and BOSA-TDA have the same results.

In the moderate workloads where CCR = 1.0, Fig. 12 demonstrates BOSA-TDA beats other approaches in all circumstances in term of makespan improvement except for in contrast with SOGA where it has not any dominance in the Cybershake graph. Also, the BOSA-TDA has superiority against other algorithms in term of monetary cost improvement in LIGO and Montage workloads, but it fails to outperform versus HEFT-TD, Lookahead, and NSGAI

Table 7 The comparison of BOSA-TDA versus state-of-the-arts in term of RPD of *speed up*

Dataset	CCR	Speedup					Relative percent deviation (RPD)			
		NSGAll	HEFT_TD	Lookahead	SOGA	BOSA-TDA	NSGAll (%)	HEFT_TD (%)	Lookahead (%)	SOGA (%)
<i>Montage</i>	0.1	1.95	1.82	1.82	1.84	1.95	0	7	7	6
<i>Montage</i>	0.5	1.31	1.06	1.00	1.31	1.31	0	24	31	0
<i>Montage</i>	1.0	0.97	0.86	0.86	0.80	1.18	22	37	37	48
<i>Epigenetics</i>	0.1	2.65	2.69	2.69	2.53	2.69	2	0	0	6
<i>Epigenetics</i>	0.5	2.00	1.70	1.70	1.41	2.00	0	18	18	42
<i>Epigenetics</i>	1.0	1.06	1.21	1.21	1.61	1.78	68	47	47	11
<i>SIPHT</i>	0.1	2.54	2.61	2.43	2.47	2.76	9	6	14	12
<i>SIPHT</i>	0.5	1.41	1.23	1.23	1.26	1.45	3	18	18	15
<i>SIPHT</i>	1.0	0.84	0.84	0.71	0.81	0.90	7	7	27	11
<i>LIGO</i>	0.1	2.20	2.00	1.97	2.10	2.20	0	10	12	5
<i>LIGO</i>	0.5	1.06	0.88	0.91	0.85	1.20	14	37	32	42
<i>LIGO</i>	1.0	0.85	0.82	0.82	0.78	1.01	19	23	23	29
<i>CyberShake</i>	0.1	2.67	2.63	2.63	2.63	2.67	0	2	2	2
<i>CyberShake</i>	0.5	1.58	1.11	1.11	1.50	1.67	6	50	50	11
<i>CyberShake</i>	1.0	1.07	0.96	0.96	1.11	1.11	4	16	16	0
<i>Average</i>							10	20	22	16

Table 8 The comparison of BOSA-TDA versus state-of-the-arts in term of RPD of *efficiency*

Dataset	CCR	Efficiency					Relative percent deviation (RPD)			
		NSGAll	HEFT_TD	Lookahead	SOGA	BOSA-TDA	NSGAll (%)	HEFT_TD (%)	Lookahead (%)	SOGA (%)
<i>Montage</i>	0.1	39.17	36.53	36.53	36.89	39.17	0	7	7	6
<i>Montage</i>	0.5	26.20	21.11	20.00	26.20	26.20	0	24	31	0
<i>Montage</i>	1.0	19.48	17.27	17.27	26.95	29.68	52	72	72	10
<i>Epigenetics</i>	0.1	66.40	67.46	67.46	63.43	67.46	2	0	0	6
<i>Epigenetics</i>	0.5	50.00	42.50	42.50	35.41	50.00	0	18	18	41
<i>Epigenetics</i>	1.0	35.41	30.35	30.35	40.47	44.73	26	47	47	11
<i>SIPHT</i>	0.1	63.59	65.31	60.92	61.96	69.04	9	6	13	11
<i>SIPHT</i>	0.5	47.15	41.13	41.13	42.02	48.33	3	18	18	15
<i>SIPHT</i>	1.0	21.01	21.01	23.86	27.23	30.20	44	44	27	11
<i>LIGO</i>	0.1	55.12	50.14	49.00	52.51	55.12	0	10	12	5
<i>LIGO</i>	0.5	26.49	21.91	22.75	21.38	30.08	14	37	32	41
<i>LIGO</i>	1.0	21.38	20.63	20.63	19.50	33.80	58	64	64	73
<i>CyberShake</i>	0.1	53.57	52.63	52.63	52.63	53.57	0	2	2	2
<i>CyberShake</i>	0.5	31.57	37.03	37.03	30.00	41.66	32	13	13	39
<i>CyberShake</i>	1.0	53.57	48.38	48.38	27.77	27.77	− 48	− 43	− 43	0
<i>Average</i>							13	21	21	18

algorithms in Epigenomics workload and against SOGA and Lookahead algorithms in SIPHT workload. In addition to, the BOSA-TDA and SOGA have the same result in the CyberShake graph. Totally, in CCR = 1.0, the BOSA-TDA has superiority in 24 cases out of 25 in term of *makespan* improvement and 19 cases out of 25 in terms of cost reduction improvement. Figure 14 is dedicated for analysis of comparative algorithms in term of SLR value.

However, in term of SLR, the BOSA-TDA beats other approaches in the majority of scenarios, but it is not dominated by the rest. In Fig. 14, for computation-intensive graphs, the BOSA-TDA has the same result in a few cases where in the most cases it has dominance against other approaches. In rather computation-intensive graphs, the BOSA-TDA has the same output with NSGAll and SOGA

in two cases; in other cases it beats others. In the moderate graphs, the BOSA-TDA beats others in all cases.

Accordingly, Fig. 15 is dedicated for comparison of comparative algorithms in term of *speedup*; it shows that the BOSA-TDA has superiority versus other approaches in term of *speedup* metric in all of the scenarios except for in some limited cases where the BOSA-TDA has the same result as the NSGAI indicates.

One of the most important metrics which releases us from the misleading conclusion is the *efficiency* metric. To this end, the Fig. 16 is dedicated for analyzing the comparison of comparative algorithms in term of *efficiency*.

Evaluation associated with the execution of different simulations points out that the BOSA-TDA dominates other algorithms in terms of *efficiency* except for in moderate Cybeshake graphs (with $CCR = 1.0$) in which three algorithms NSGAI, HEFT-TD, and Lookahead have dominance against BOSA-TDA. In the other words, the BOSA-TDA dominates in 22 cases out of 25 cases versus other approaches. As Fig. 16 depicts, in $CCR = 1.0$ scenarios, the NSGA II competes with the BOSA-TDA in terms of evaluation metrics only in some datasets, but in other cases, the BOSA-TDA outperforms other state-of-the-arts significantly.

Data analysis

This section presents data analysis to a better understanding of the proposed algorithm's performance in contrast with other approaches in terms of prominent metrics derived from literature. In addition, a relative percentage deviation (RPD) metric is applied to point out the amount of enhancement gained via proposed BOSA-TDA approach [1]. To do so, Tables 4, 5, and 6 are dedicated to this reason. Table 4 illustrates a *makespan* comparison of different comparative algorithms. To have a ranking list in terms of *makespan* metric from the best to the worst algorithm, NSGAI, SOGA, HEFT-TD, and Lookahead algorithms are placed in the ranking list, but after the BOSA-TDA. The negative cell value means deterioration whereas the zero value means no improvement was gained.

Table 5 illustrates the *cost reduction* comparison of different algorithms. For cost reduction metrics, Lookahead, HEFT-TD, NSGAI, and SOGA algorithms are placed in the ranking list from the best to the worst, but after the BOSA-TDA which is in the first place.

In this regard, Table 6 illustrates the *SLR* metric comparison of different comparative algorithms. For *SLR* metric, NSGAI, SOGA, HEFT-TD, and Lookahead algorithms are placed in the ranking list from the best to the worst, but after the proposed BOSA-TDA.

Table 7 illustrates the *Speedup* metric comparison of different comparative algorithms. For *Speedup* metric, again NSGAI, SOGA, HEFT-TD, and Lookahead algorithms are

placed in the ranking list from the best to the worst, but after BOSA-TDA.

Table 8 illustrates the *Efficiency* metric comparison of different comparative algorithms. Also, for the *efficiency* metric, NSGAI, SOGA, HEFT-TD, and Lookahead algorithms are placed in the ranking list from the best to the worst, but after BOSA-TDA.

Conclusion and future work

The majority of existing workload scheduling algorithms in the cloud environment only intend to minimize *makespan* metric whereas they neglect to consider user service bill. Since cloud providers provision different processing power in terms of VM configurations, it burdens variable charges for subscribers. This is the reason that this article formulated workload scheduling in hybrid cloud architecture as a bi-objective optimization problem. To deal with the combinatorial problem, a novel hybrid population-based simulated annealing algorithm by applying the duplication technique has been presented which is named BOSA-TDA. To evaluate the performance of BOSA-TDA in terms of derived prominent metrics from literature, extensive scenarios on set of well-known workloads have been conducted. The reported results from the simulation of extensive scenarios proved the superiority of the proposed algorithm against other state-of-the-art approaches in terms of metrics in this ambit. For future work, we intend to model cloud reliability for mission-critical workloads; then, the scheduling problems can be formulated as a new bi-objective optimization algorithm with total execution time and reliability perspectives.

Funding No fund available.

Declarations

Conflict of interest The author declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Hosseini Shirvani M (2020) A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems. *Eng Appl Artif Intell* 90(December 2019):103501. <https://doi.org/10.1016/j.engappai.2020.103501>
- Hosseini Shirvani M, Gorji AB (2020) Optimization of automatic web services composition using genetic algorithm. *Int J Cloud Comput* 9(4):397–411. <https://doi.org/10.1504/IJCC.2020.112313>
- Hosseini Shirvani M (2020) To move or not to move: an iterative four-phase cloud adoption decision model for IT outsourcing based On TCO. *J Soft Comput Inf Technol* 9(1):7–17
- Masdari M, ValiKardan S, Shahi Z, Azar I (2016) Towards workflow scheduling in cloud computing. *J Netw Comput Appl* 66:64–82. <https://doi.org/10.1016/J.JNCA.2016.01.018>
- Bharathi S, Chervenak A, Deelman E, Mehta G, Su M-H and Vahi K (2008) Characterization of scientific workflows. In: 2008 third workshop on workflows in support of large-scale science. pp 1–10, <https://doi.org/10.1109/WORKS.2008.4723958>
- Masdari M, Khoshnevis A (2019) A survey and classification of the workload forecasting methods in cloud computing. *Clust Comput* 23(4):2399–2424. <https://doi.org/10.1007/S10586-019-03010-3>
- Deelman E et al (2015) Pegasus, a workflow management system for science automation. *Futur Gener Comput Syst* 46:17–35. <https://doi.org/10.1016/j.future.2014.10.008>
- Hosseini Shirvani MS (2018) A new shuffled genetic-based task scheduling algorithm in heterogeneous distributed systems. *J Adv Comput Res* 9(4):19–36. http://jacr.iausari.ac.ir/article_660143.html
- Lu Y-C et al (2018) Service deployment and scheduling for improving performance of composite cloud services. *Comput Electr Eng*. <https://doi.org/10.1016/j.compeleceng.2018.07.018>
- Safari M, Khorsand R (2018) Energy-aware scheduling algorithm for time-constrained workflow tasks in DVFS-enabled cloud environment. *Simul Model Pract Theory* 87:311–326. <https://doi.org/10.1016/j.simpat.2018.07.006>
- Utrera G, Farreras M, Fornes J (2019) Task packing: efficient task scheduling in unbalanced parallel programs to maximize CPU utilization. *J Parallel Distrib Comput* 134:37–49. <https://doi.org/10.1016/j.jpdc.2019.08.003>
- Durillo JJ, Nae V, Prodan R (2014) Multi-objective energy-efficient workflow scheduling using list-based heuristics. *Futur Gener Comput Syst* 36:221–236. <https://doi.org/10.1016/j.future.2013.07.005>
- Lin C-S, Lin C-S, Lin Y-S, Hsiung P-A, Shih C (2013) Multi-objective exploitation of pipeline parallelism using clustering, replication and duplication in embedded multi-core systems. *J Syst Archit* 59(10):1083–1094. <https://doi.org/10.1016/j.sysarc.2013.05.024>
- Tang Q, Zhu L-H, Zhou L, Xiong J, Wei J-B (2020) Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems. *J Parallel Distrib Comput* 138:115–127. <https://doi.org/10.1016/j.jpdc.2019.12.012>
- Țigănoaia B, Iordache G, Negru C, Pop F (2019) Scheduling in CloudSim of interdependent tasks for SLA design. *Stud Inform Control* 28:477–484. <https://doi.org/10.24846/v28i4y201911>
- Wu M-Y, Gajski DD (1990) Hypertool: a programming aid for message-passing systems. *IEEE Trans Parallel Distrib Syst* 1(3):330–343. <https://doi.org/10.1109/71.80160>
- Kwok Y-K, Ahmad I (1996) Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Trans Parallel Distrib Syst* 7(5):506–521. <https://doi.org/10.1109/71.503776>
- Sih GC, Lee EA (1993) A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans Parallel Distrib Syst* 4:175–187. <https://doi.org/10.1109/71.207593>
- Topcuoglu H, Hariri S, Wu MY (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans Parallel Distrib Syst* 13(3):260–274. <https://doi.org/10.1109/71.993206>
- Daoud MI, Kharma N (2008) A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *J Parallel Distrib Comput* 68(4):399–409. <https://doi.org/10.1016/j.jpdc.2007.05.015>
- Guo P and Xue Z (2017) Cost-effective fault-tolerant scheduling algorithm for real-time tasks in cloud systems. In: International conference on communication technology proceedings, ICCT, 2018, pp 1942–1946, <https://doi.org/10.1109/ICCT.2017.8359968>
- Tong Z, Deng X, Chen H, Mei J, Liu H (2020) QL-HEFT: a novel machine learning scheduling scheme base on cloud computing environment. *Neural Comput Appl* 32(10):5553–5570. <https://doi.org/10.1007/s00521-019-04118-8>
- Liou J and Palis MA (1996) An efficient task clustering heuristic for scheduling DAGs on multiprocessors. In: Symp. Parallel Distrib. Process., no. February, pp 152–156
- Al-Rahayfeh A, Atiewi S, Abuhusseini A, Almiani M (2019) Novel approach to task scheduling and load balancing using the dominant sequence clustering and mean shift clustering algorithms. *Futur Internet* 11(5):109. <https://doi.org/10.3390/fi11050109>
- Kruatrachue B, Lewis T (1988) Grain size determination for parallel processing. *IEEE Softw*. <https://doi.org/10.1109/52.1991>
- Ahmad I, Kwok Y-K (1998) On exploiting task duplication in parallel program scheduling. *IEEE Trans Parallel Distrib Syst* 9(9):872–892. <https://doi.org/10.1109/71.722221>
- Kruatrachue B, Lewis TG (1987) Duplication scheduling heuristics (dsh): a new precedence task scheduler for parallel processor systems. Oregon State Univ, Corvallis
- Mishra PK, Mishra A, Mishra KS, Tripathi AK (2012) Benchmarking the clustering algorithms for multiprocessor environments using dynamic priority of modules. *Appl Math Model*. <https://doi.org/10.1016/j.apm.2012.02.011>
- Ranaweera S, Agrawal DP (2000) A scalable task duplication based scheduling algorithm for heterogeneous systems. *Proc Int Conf Parallel Process*. <https://doi.org/10.1109/ICPP.2000.876154>
- Lopes Genez TA, Sakellariou R, Bittencourt LF, Mauro Madeira ER and Braun T (2018) Scheduling scientific workflows on clouds using a task duplication approach. In: 2018 IEEE/ACM 11th international conference on utility and cloud computing (UCC). pp 83–92, <https://doi.org/10.1109/UCC.2018.00017>
- Behnamian J, Fatemi Ghomi SMT (2013) The heterogeneous multi-factory production network scheduling with adaptive communication policy and parallel machine. *Inf Sci (Ny)*. <https://doi.org/10.1016/j.ins.2012.07.020>
- Lu H, Niu R, Liu J, Zhu Z (2013) A chaotic non-dominated sorting genetic algorithm for the multi-objective automatic test task scheduling problem. *Appl Soft Comput J*. <https://doi.org/10.1016/j.asoc.2012.10.001>
- Świecicka A, Seredynski F, Zomaya AY (2006) Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support. *IEEE Trans Parallel Distrib Syst*. <https://doi.org/10.1109/TPDS.2006.38>
- Zomaya AY, Ward C, Macey B (1999) Genetic scheduling for parallel processor systems: comparative studies and performance issues. *IEEE Trans Parallel Distrib Syst*. <https://doi.org/10.1109/71.790598>
- Wang J, Duan Q, Jiang Y and Zhu X (2010) A new algorithm for grid independent task schedule: genetic simulated annealing. In: 2010 World Automation Congress, WAC 2010
- Torres-Jimenez J and Rodriguez-Tello E (2010) Simulated annealing for constructing binary covering arrays of variable strength.

- In: 2010 IEEE world congress on computational intelligence, WCCI 2010-2010 IEEE congress on evolutionary computation, CEC 2010. <https://doi.org/10.1109/CEC.2010.5586148>
37. Noorian Talouki R, Hosseini Shirvani M, Motameni H (2021) A hybrid meta-heuristic scheduler algorithm for optimization of workflow scheduling in cloud heterogeneous computing environment. *J Eng Des Technol Emerald Publ.* <https://doi.org/10.1108/JEDT-11-2020-0474>.
 38. Tanha M, Hosseini Shirvani M, Rahmani AM (2021) A hybrid metaheuristic task scheduling algorithm based on genetic and thermodynamic simulated annealing algorithms in cloud computing environments. *Neural Comput & Applic.* <https://doi.org/10.1007/s00521-021-06289-9>
 39. Guo L, Zhao S, Shen S, Jiang C (2012) Task scheduling optimization in cloud computing based on heuristic Algorithm. *J Networks.* <https://doi.org/10.4304/jnw.7.3.547-553>
 40. Zhao F, Tang J, Wang J, Jonrinaldi (2014) An improved particle swarm optimization with decline disturbance index (DDPSO) for multi-objective job-shop scheduling problem. *Comput Oper Res.* <https://doi.org/10.1016/j.cor.2013.11.019>
 41. Pandey S, Wu L, Guru SM and Buyya R (2010) A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: Proceedings—international conference on advanced information networking and applications, AINA. <https://doi.org/10.1109/AINA.2010.31>.
 42. Wang J, Li F, Chen A (2012) An improved PSO based task scheduling algorithm for cloud storage system. *Adv Inf Sci Serv Sci.* <https://doi.org/10.4156/AISS.vol4.issue18.57>
 43. Li H, Wang L and Liu J (2010) Task scheduling of computational grid based on particle swarm algorithm. In: 3rd international joint conference on computational sciences and optimization, CSO 2010: theoretical development and engineering practice. <https://doi.org/10.1109/CSO.2010.34>
 44. Feng M, Wang X, Zhang Y, and Li J (2013) Multi-objective particle swarm optimization for resource allocation in cloud computing. In: Proceedings—2012 IEEE 2nd international conference on cloud computing and intelligence systems. IEEE CCIS 2012 <https://doi.org/10.1109/CCIS.2012.6664566>
 45. Delavar AG (2012) Task scheduling in grid environment with ant colony method for cost and time. *Int J Comput Sci Eng Appl.* <https://doi.org/10.5121/ijcsea.2012.2501>
 46. Khambre PD, Deshpande A, Mehta A, Sain A (2014) Modified pheromone update rule to implement ant colony optimization algorithm for workflow scheduling algorithm problem in grids. *Int J Adv Res Comput Sci Technol* 2(2):424–429
 47. Ferrandi F, Lanzi PL, Pilato C, Sciuto D, Tumeo A (2010) Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. *IEEE Trans Comput Des Integr Circuits Syst.* <https://doi.org/10.1109/TCAD.2010.2048354>
 48. Lin M, Xi J, Bai W, Wu J (2019) Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. *IEEE Access.* <https://doi.org/10.1109/ACCESS.2019.2924414>
 49. Srikanth GU, Maheswari VU, Shanthi P, Siromoney A (2012) Tasks scheduling using ant colony optimization. *J Comput Sci.* <https://doi.org/10.3844/jcssp.2012.1314.1320>
 50. Kong X, Xu J, Zhang W (2015) Ant colony algorithm of multi-objective optimization for dynamic grid scheduling. *Metall Min Ind* 1(3):236–243
 51. Idris H, Ezugwu AE, Junaidu SB, Adewumi AO (2017) An improved ant colony optimization algorithm with fault tolerance for job scheduling in grid computing systems. *PLoS One.* <https://doi.org/10.1371/journal.pone.0177567>
 52. Yildiz AR (2013) Optimization of cutting parameters in multi-pass turning using artificial bee colony-based approach. *Inf Sci.* <https://doi.org/10.1016/j.ins.2012.07.012>
 53. Yildiz AR (2013) Cuckoo search algorithm for the selection of optimal machining parameters in milling operations. *Int J Adv Manuf Technol.* <https://doi.org/10.1007/s00170-012-4013-7>
 54. Arabnejad H, Barbosa JG (2014) List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans Parallel Distrib Syst* 25(3):682–694. <https://doi.org/10.1109/TPDS.2013.57>
 55. Khan M (2012) Scheduling for heterogeneous Systems using constrained critical paths. *Parallel Comput* 38:175–193. <https://doi.org/10.1016/j.parco.2012.01.001>
 56. Thaman J, Singh M (2017) Green cloud environment by using robust planning algorithm. *Egypt Inform J* 18(3):205–214. <https://doi.org/10.1016/j.eij.2017.02.001>
 57. Gkoutioudi K, Karatza HD (2010) Task cluster scheduling in a grid system. *Simul Model Pract Theory* 18(9):1242–1252. <https://doi.org/10.1016/j.simpat.2010.04.011>
 58. Amini A, Wah TY, Saybani MR and Yazdi SRAS (2011) A study of density-grid based clustering algorithms on data streams. In: Proceedings—2011 8th international conference on fuzzy systems and knowledge discovery, FSKD 2011. <https://doi.org/10.1109/FSKD.2011.6019867>
 59. Shin KS, Cha MJ, Jang MS, Jung JH, Yoon WO, Choi SB (2008) Task scheduling algorithm using minimized duplications in homogeneous systems. *J Parallel Distrib Comput.* <https://doi.org/10.1016/j.jpdc.2008.04.001>
 60. Sinnen O, To A, Kaur M (2011) Contention-aware scheduling with task duplication. *J Parallel Distrib Comput* 71(1):77–86. <https://doi.org/10.1016/j.jpdc.2010.10.004>
 61. Tang X, Li K, Liao G, Li R (2010) List scheduling with duplication for heterogeneous computing systems. *J Parallel Distrib Comput* 70(4):323–329. <https://doi.org/10.1016/j.jpdc.2010.01.003>
 62. Ahmad I and Kwok Y (1994) A new approach to scheduling parallel programs using task duplication. In: 1994 International Conference on Parallel Processing (ICPP'94). pp 47–51, <https://doi.org/10.1109/ICPP.1994.37>
 63. Zhang Y, Zhou J, Sun J (2019) Scheduling bag-of-tasks applications on hybrid clouds under due date constraints. *J Syst Arch* 101:101654. <https://doi.org/10.1016/j.sysarc.2019.101654>
 64. Garai G, Chaudhuri BB (2013) A novel hybrid genetic algorithm with Tabu search for optimizing multi-dimensional functions and point pattern recognition. *Inf Sci (Ny).* <https://doi.org/10.1016/j.ins.2012.09.012>
 65. Akbari M, Rashidi H, Alizadeh SH (2017) An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems. *Eng Appl Artif Intell* 61(February):35–46. <https://doi.org/10.1016/j.engappai.2017.02.013>
 66. Kim H, Kang S (2011) Communication-aware task scheduling and voltage selection for total energy minimization in a multiprocessor system using ant colony optimization. *Inf Sci (Ny).* <https://doi.org/10.1016/j.ins.2011.04.037>
 67. Jena RK (2017) Task scheduling in cloud environment: a multi-objective ABC framework. *J Inf Optim Sci.* <https://doi.org/10.1080/02522667.2016.1250460>
 68. Li Y, Wang C, Gao L, Song Y, Li X (2020) An improved simulated annealing algorithm based on residual network for permutation flow shop scheduling. *Complex Intell Syst.* <https://doi.org/10.1007/s40747-020-00205-9>
 69. Zhao F, Hu X, Wang L, Li Z (2021) A memetic discrete differential evolution algorithm for the distributed permutation flow shop scheduling problem. *Complex Intell Syst.* <https://doi.org/10.1007/s40747-021-00354-5>
 70. Shanmugapriya R, Padmavathi S and Shalinie SM (2019) Contention awareness in task scheduling using tabu search. In: 2009

- IEEE International Advance Computing Conference, IACC 2009. <https://doi.org/10.1109/IADCC.2009.4809020>.
71. Xu Y, Li K, Hu J, Li K (2014) A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Inf Sci (Ny)* 270:255–287. <https://doi.org/10.1016/j.ins.2014.02.122>
 72. Sandokji S, Eassa F (2019) Dynamic variant rank HEFT task scheduling algorithm toward exascale computing. *Procedia Comput Sci* 163:482–493. <https://doi.org/10.1016/j.procs.2019.12.131>
 73. Tasoglu G, Yildiz G (2019) Simulated annealing based simulation optimization method for solving integrated berth allocation and quay crane scheduling problems. *Simul Model Pract Theory* 97:101948. <https://doi.org/10.1016/j.simpat.2019.101948>
 74. Medhat AT, Ashraf E-S, Keshk Arabi E and Fawzy AT (2013) An ant algorithm for cloud task scheduling. In: Proceedings of the 1st international workshop on cloud computing and information security. <https://doi.org/10.2991/ccis-13.2013.40>
 75. Jafari Navimipour N, Sharifi Milani F (2015) Task scheduling in the cloud computing based on the cuckoo search algorithm. *Int J Model Optim* 5(1):44–47. <https://doi.org/10.7763/IJMO.2015.V5.434>
 76. Shirvani MH, Amirsoleimani N, Salimpour S, Azab A (2017) Multi-criteria task scheduling in distributed systems based on fuzzy TOPSIS. *Can Conf Elect Comput Eng*. <https://doi.org/10.1109/CCECE.2017.7946721>
 77. Zhang L, Li KK, Li C, Li KK (2016) Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. *Inf Sci (Ny)* 7:1–16. <https://doi.org/10.1016/j.ins.2016.08.003>
 78. Vecchi MP, Kirkpatrick S, Gelatt CD (1983) Optimization by simulated annealing. *Science* (80–) 44(8):085201. <https://doi.org/10.1088/1751-8113/44/8/085201>
 79. Damodaran P, Vélez-Gallego MC (2012) A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert Syst Appl* 39(1):1451–1458. <https://doi.org/10.1016/j.eswa.2011.08.029>
 80. Farzai S, Shirvani MH, Rabbani M (2020) Multi-objective communication-aware optimization for virtual machine placement in cloud datacenters. *Sustain Comput Inform Syst*. <https://doi.org/10.1016/j.suscom.2020.100374>
 81. Deb K, Agrawal S, Pratap A, Meyarivan T (2000) A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*, vol 1917. Springer, Berlin, pp 849–858
 82. Fan J, Zhao L, Du L, Zheng Y, and Science C (2010) Crowding-distance-based multi-objective particle swarm optimization. 218–225
 83. Černý V (1985) Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J Optim Theory Appl* 45(1):41–51. <https://doi.org/10.1007/BF00940812>
 84. Lundy M, Mees A (1986) Convergence of an annealing algorithm. *Math Program* 34(1):111–124. <https://doi.org/10.1007/BF01582166>
 85. Dowsland KA, Thompson JM (2012) Simulated annealing. In: Rozenberg G, Bäck T, Kok JN (eds) *Handbook of natural computing*, vol 1–4. Heidelberg, Springer, pp 1623–1655
 86. Masdari M, Zangakani M (2019) Efficient task and workflow scheduling in inter-cloud environments: challenges and opportunities. *J Supercomput* 76(1):499–535. <https://doi.org/10.1007/s11227-019-03038-7>
 87. Durillo JJ, Prodan R (2014) Multi-objective workflow scheduling in Amazon EC2. *Cluster Comput* 17(2):169–189. <https://doi.org/10.1007/s10586-013-0325-0>
 88. Haidri RA, Katti CP, Saxena PC (2019) Cost-effective deadline-aware stochastic scheduling strategy for workflow applications on virtual machines in cloud computing. *Concurr Comput Pract Exp* 31(7):e5006. <https://doi.org/10.1002/cpe.5006>
 89. “Microsoft Azure.” Available: <https://azure.microsoft.com/en-us/pricing/>. [Accessed: 17-Jan-2021]
 90. “Salesforce.” Available: <https://www.salesforce.com/editions-pricing/overview>. [Accessed: 17-Jan-2021]
 91. “Google.” Available: <https://cloud.google.com/compute/all-pricing>. [Accessed: 17-Jan-2021]
 92. “Amazon EMR pricing.” Available: <https://aws.amazon.com/emr/pricing/>. [Accessed: 03-Mar-2020]
 93. “Amazon EC2 Instance Types.” Available: <https://aws.amazon.com/ec2/instance-types/>. [Accessed: 03-Mar-2020]
 94. Arabnejad V, Bubendorfer K, Ng B (2019) Dynamic multi-workflow scheduling: a deadline and cost-aware approach for commercial clouds. *Futur Gener Comput Syst* 100:98–108. <https://doi.org/10.1016/j.future.2019.04.029>
 95. Zhou J, Wang T, Cong P, Lu P, Wei T, Chen M (2019) Cost and makespan-aware workflow scheduling in hybrid clouds. *J Syst Archit* 100:101631. <https://doi.org/10.1016/j.sysarc.2019.08.004>
 96. Bittencourt LF, Mauro Madeira ER (2011) HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds. *J Serv Internet Appl* 2(3):207–227. <https://doi.org/10.1007/s13174-011-0032-0>
 97. Wang G, Wang Y, Liu H, Guo H (2016) HSIP: a novel task scheduling algorithm for heterogeneous computing. *Sci Programm* 2016:1–11

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.