Cluster Computing
https://doi.org/10.1007/s10586-020-03145-8

DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing

Amir Iranmanesh[1] · Hamid Reza Naji[1]

Received: 20 January 2020 / Revised: 10 June 2020 / Accepted: 21 June 2020

Abstract
Cloud infrastructures are suitable environments for processing large scientific workflows. Nowadays, new challenges are emerging in the field of optimizing workflows such that it can meet user's service quality requirements. The key to workflow optimization is the scheduling of workflow tasks, which is a famous NP-hard problem. Although several methods have been proposed based on the genetic algorithm for task scheduling in clouds, our proposed method is more efficient than other proposed methods due to the use of new genetic operators as well as modified genetic operators and the use of load balancing routine. Moreover, a solution obtained from a heuristic used as one of the initial population chromosomes and an efficient routine also used for generating the rest of the primary population chromosomes. An adaptive fitness function is used that takes into account both cost and makespan. The algorithm introduced in this paper utilizes a load balancing routine to maximize resources' efficiency at execution time. The performance of the proposed algorithm is evaluated by comparing the results with state of the art algorithms of this field, and the results indicate that the proposed algorithm has remarkable superiority in comparison to other algorithms and performs task scheduling with the least makespan and cost.

Keywords Cloud computing · Genetic algorithm · Heuristic · Workflow scheduling

1 Introduction

Nowadays scientific community is confronted with new challenges in terms of experiments and simulations of big data. The heterogeneity of the scientific applications and the execution platforms have added these challenges in the management of large-scale data [1]. These scientific workflows that include big data require computations and environments for high-performance processing. Cloud computing is an example of distributed computing that is suitable for the efficient handling of big data, as well as dynamic and distributed services through the Internet [2]. Types of services provided by cloud computing include Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) [3].

Cloud computing delivers dynamic and high scalability resources to users in the same form as water and electricity provided to households these days. Workflow scheduling in the cloud system is the mapping of tasks to computational resources for execution such that the priorities of the tasks execution are maintained [4]. There are two phases in the workflow scheduling in the cloud computing environment. The first phase is mapping the tasks to the resources, and the second phase is the execution of tasks in a single resource [5]. In this paper, task scheduling is performed using an intelligent algorithm. Optimization objectives for workflow scheduling in the cloud computing environment are including makespan, cost, throughput, and load balancing.

The deadline is the time constraint defined by the users for workflow execution. The deadline constrained scheduling problem is to map every task T_i onto a suitable processor P_i to minimize the execution cost of the workflow and complete it within the deadline. Therefore,

✉ Amir Iranmanesh
a.iranmanesh@student.kgut.ac.ir

Hamid Reza Naji
h.naji@kgut.ac.ir

[1] Faculty of Electronic and Computer Engineering, Department of Computer Engineering and Information Technology, Graduate University of Advanced Technology, Kerman, Iran

Published online: 29 June 2020

this paper aims to find the optimal mapping between each workflow task and the available cloud resources such that makespan and financial costs are reduced simultaneously. Moreover, the deadline is also met.

Recently, many algorithms for task scheduling in the cloud environment have been introduced. Yuan et al. proposed the Green Cloud Data Centers (GCDCs) [6]. In this multi-objective approach, the authors used spatial differences and decomposition to establish an efficient tradeoff between cost and time to process all tasks. However, the initial population is randomly generated in this method. It also does not take into account the balanced workload distribution between processors. Simultaneously, in our proposed algorithm, besides high simplicity and flexibility, new and modified operators have been used in the genetic algorithm, which significantly reduces the time to converge to the optimal solution. Moreover, the load distribution routine is also used to increase the performance of the processors.

As such, the algorithm introduced in [7] for task scheduling in cloud computing environment used a random population that significantly reduces the convergence speed of the algorithm. Therefore, we decided to design an algorithm that would have a higher speed to achieve the optimal solution. In this paper, the convergence rate of the genetic algorithm has been improved through using a well-known heuristic, and the initial population is generated by portioning workflow tasks and property of critical path, and an acceptable solution is obtained at a reasonable time.

The rest of the paper is organized as follows. Section 2 introduces the background and summarizes the related work, and Sect. 3 describes the task scheduling model. The proposed algorithm is described in Sect. 4. Section 5 presents the performance analysis and discussion. The paper concluded in Sect. 6.

2 Background

In the case of Quality-of-Service (QoS) constraint scheduling, most of the available algorithms will improve one of the QoS parameters. These two parameters are the user-defined deadline and financial constraint. Liu et al. introduced a genetic algorithm that uses an adaptive cost function to satisfy constraints [8]. This algorithm also uses a co-evolutionary method to adjust the rate of crossover and mutation, which increases the rate of convergence. Wang et al. used a predictive genetic algorithm to predict future resources and reduce the makespan [9]. Liu et al. introduced a Multiple-Priority Queues Genetic Algorithm (MPQGA) [10], which uses multiple queues for sorting the tasks, and implements a genetic algorithm according to a list of task priorities. The initial population consists of multiple queues of the tasks, and the tasks are mapped to processors based on their lowest finish time. Algorithms such as particle swarm optimization (PSO) [11], ant colony optimization (ACO) [12], artificial bee colony algorithm (ABC) [13], and genetic algorithm (GA) [7] [16, 17] are practical algorithms for different scheduling problems. These algorithms require sufficient samples of the candidate solutions in the search space. However, the most significant drawback of basic genetic algorithms is evident when the search space is largely due to the high computational cost required.

Existing task scheduling can be divided into two general categories: (1) methods that strive to optimize energy consumption. (2) methods focusing on optimizing workload processing and improving service quality parameters.

2.1 Energy optimization techniques

Wang et al. introduced a task scheduling framework that considered deadline, data localization, and resource efficiency enhancements to reduce energy consumption in a heterogeneous cloud [14]. The proposed framework includes three sections on creating lists of tasks, tasks scheduling, and updating slot lists. In terms of deadline constraints, the number of slots allocated to the jobs, the possible processing times of the jobs, and a new logical sequence of tasks has been proposed. Tasks are assigned to high-quality slots from local servers, clustered servers, and remote servers promptly to improve data locality. Available clusters have been proposed that find not only the available slots but also enhance the efficiency of server resources using fuzzy logic.

Sun et al. proposed an efficient task scheduling for the fog-cloud environment [15]. The authors consider the mobility of cloud nodes as the most influential factor in the scheduling of tasks. The authors modeled the distribution of resources among clusters in order to maximize resource efficiency and the number of tasks that can be performed in the cloud environment. However, this approach did not consider communication-based tasks.

Tang et al. introduced the efficient energy scheduling algorithm in the cloud environment [16]. This algorithm reduces the idle time of the processors (employs inefficient processors.) Their proposed algorithm first performs initial scheduling by randomly assigning tasks to the processors and then calculates the overall makespan and deadline based on the HEFT heuristic.

There are lots of works done on Virtual Machine (VM) or task scheduling, VM placement, and resource allocation in the cloud environment. Yadav et al. introduced an energy-based scheduling algorithm in a cloud computing environment that selects VMs according to the Service-Level Agreement (SLA) to prevent hosts from overloading

[17]. The authors used three adaptive and consistent models to minimize energy consumption and SLA violations. This method uses the overloaded host identification algorithm and the VM selection of the overloaded host to increase efficiency and reduce energy consumption.

Yadav et al. also introduced an energy-aware scheduling algorithm to reduce energy at a cloud data center [18]. This algorithm identifies overloaded hosts and selects VMs to combine VM in under-loaded hosts. After all, resources have been allocated to all VM; underload hosts should be in energy storage mode to reduce energy consumption. Moreover, the authors used a regression-based algorithm to determine the threshold of CPU performance to identify overloaded hosts. All in all, our proposed VM consolidation algorithm is based on the study of li [19].

2.2 Workload processing optimization techniques

Garg et al. introduced a scheduling algorithm in the cloud environment that simultaneously increases reliability and reduces energy consumption [20]. The algorithm consists of four levels; the first level is a topology ordered by workflow tasks. At the second level, it used a heuristic for task clustering to reduce the cost of communication between tasks. In the third level, user-defined QoS constraints are applied, and in the last step, the tasks are assigned to the appropriate processors according to their required voltage levels. However, due to its heuristic nature, the performance of this algorithm is significantly reduced in heterogeneous cloud systems with large workflows and is unable to provide the fully guaranteed QoS parameters required by users. Specifically, our algorithm, due to the combination of advanced genetic algorithm and a robust heuristic optimizes the two most critical performance parameters, which means time and cost.

Arabnejad et al. proposed a scheduling algorithm, Proportional Deadline Constrained (PDC), that solved the problem of scientific workflow scheduling in the cloud environment [21]. The primary purpose of PDC is to optimize makespan by considering the deadline constraint. PDC consists of multiple phases: the first phase is the leveling of the workflow. The second phase is the proportional deadline distribution to the levels. The third phase is the selection of the best instance based on the least time and cost. However, this algorithm is heuristic-based and does not consider the load balancing of processors.

Keshanchi et al. introduced the N-GA algorithm for static task scheduling in a cloud computing environment [22]. The proposed algorithm combines the benefits of the classic genetic algorithm and the HEFT heuristic. However, in this algorithm, the initial population is also generated randomly. Moreover, this method does not consider the Deadline constraint, and only the makespan parameter is considered, and other QoS parameters, including cost, are not considered.

Ghobaei et al. introduced the Task Scheduling algorithm based on Moth-Flame Optimization (TS-MFO) [23]. In TS-MFO, a set of optimal tasks are assigned to FOG nodes so that the optimal makespan is required to perform these tasks efficiently. In this algorithm, the initial population is optimized using a method based on the MFO that considers moths' behavior at night around the flame to assign tasks to FOG nodes to reduce the average time of task execution. However, TS-MFO considers only the makespan parameter; therefore, the cost parameter is not significant in this method.

Li et al. introduced a PSO-based scheduling algorithm for interdependent tasks in a local grid network [11]. First, a scheduling model of the computational grid is created, then the particle swarm algorithm is changed in continuous space searching to the integer space searching. Selects the appropriate inertial weight value and improves the algorithm's search capacity. kimpan et al. introduced a load-balanced distribution between VMs in the cloud computing environment [13]. The hybridization of ABC with robust heuristic has been used to reduce makespan. Also, in this method, tasks are considered randomly divided into two categories without considering the types of data distribution.

Basu et al. introduced an intelligent model for task scheduling of IoT applications, named the Genetic Algorithm Ant Colony Optimization (GAACO). GAACO is implemented by combining the Genetic Algorithm (GA) and Ant Colony Optimization algorithm (ACO) [12]. Specifically, GAACO speeds up the convergence of the algorithm to solve task scheduling problems. However, the major disadvantage of this model is complex and significant space problems because the algorithm is stuck at the local optima. However, DCHG-TS also works in complicated and broad problems due to the use of modified operators and new addition operators to the basic genetic algorithm.

Mortazavi-Dehkordi et al. introduced an efficient scheduling framework that considered the deadline limit defined in Big Data applications of the public cloud environment [24]. This scheduling model considers the cost of increasing the public cloud's efficiency, limiting the deadline meeting, and reducing the delay for workflows processing. This algorithm uses two procedures, one is partitioning, and the other is replication.

Kaur et al. introduced the multi-objective scheduling method, named the Predict Genetic Algorithm (PGA) [25]. In this method, the classical genetic algorithm was combined with robust Predict Earliest Finish Time (PEFT) heuristic. PGA used the solution of PEFT as one

chromosome of the initial population to increase the convergence rate. However, the authors used the classic genetic algorithm without increasing efficiency. Moreover, the machines' performance during operation has not been taken into account.

A recently proposed GA, a Cost optimization of Workflow Task Scheduling based on the Genetic Algorithm (CWTS-GA) [7], considers the dynamic elastic characteristics of the cloud resources provisions. Still, a disadvantage is that it takes a long time to converge into an optimal solution due to the use of the near-random initial population. In this algorithm, priority is assigned to each task by the up-bottom partitioning method and uses two dimensions encoding method to schedule the tasks. Moreover, the authors designed the fitness function in such a way that it takes into account both cost and scheduling length. In summary, we use two-fold genetic operators (mutation and crossover). The experimental results show the superiority of our algorithm over CWTS-GA and state of the art algorithms in terms of the performance and also, using the load balancing routine is another advantage of our proposed algorithm at runtime.

3 Scheduling model

In this section, the heterogeneous model, applications, and used system are described.

3.1 System model

In this paper, cloud resources are defined as heterogeneous processors that are fully connected through a high-speed network, as shown in Fig. 1. A set of tasks in the Directed Acyclic Graph (DAG) can be executed on a single processor at any time. If dependent tasks are scheduled on two different processors, the communication time between two dependent tasks should be considered. Table 1 represents a list of notations and definitions used in this paper.

3.2 Application model

In this paper, the workflow application is depicted as a DAG. In this graph, vertices represent the tasks, and the edges between the vertices represent the execution priority of the tasks, which means that (t_i, t_j) describes that task t_j does not process unless processing of task t_i is complete. In this case, the task t_i is called the parent task of t_j, and the task t_j is called the child task of t_i. A set of input and output tasks is considered in the DAG of this paper. Any task without a parent is called as the entry task t_{entry}, and a task with no child task is called the exit task t_{exit} [26]. Figure 2 shows a DAG that is generated randomly. In the workflow graph considered in this paper, it is assumed that when the processing of each task is completed, its output is guided to the next task. Due to the Depth First Search (DFS) forest, corresponding to the directional graph of Fig. 2 has no back edge, so this graph is definitely without a cycle.

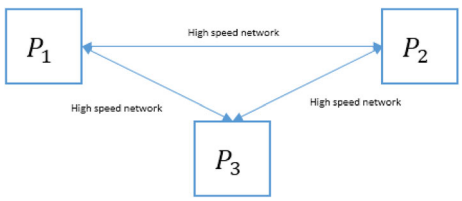


Fig. 1 A fully connected parallel system with 3 processors

As shown in Fig. 2, the DAG of application has two entry tasks t_0, t_1, and four exit tasks t_{12}, t_{13}, t_{14}, and t_{15}.

3.3 Task leveling

Eq. (1) calculates the $down-top$ rank of each of the tasks, and descending order of the corresponding $down-top$ rank values generates a priority list of tasks.

$$Rank_{down-top}(t_i) = wt_i + max_{t_j \in SUCC(t_i)}(c(t_i, t_j) + Rank_{down-top}(t_i)) \quad (1)$$

Where $w(t_i)$ is the amount of time required to process the task on the execution node,$c(t_i, t_j)$ is the cost of data transferring from a parent node t_i to a child node t_j in a workflow, and $SUCC(t_i)$ is the set of all successor tasks that are immediately after t_i. The $Rank_{down-top}$ is obtained recursively by traversing the application DAG upward, starting from the exit task [27].

The whole scheduling is affected by a critical path. Specifically, a path from the entry node to the exit node with the highest costs of edges and vertices is called the critical path. Eq. (2) calculates the Level of the Task (LT).

$$LT = Rank_{down-top}(t_i) + max_{t_j \in pred(t_i)}\left(\bar{W}_j + \bar{C}(t_i, t_j) + Rank_{down-top}(t_i)\right) \quad (2)$$

In this paper, significant contributions are made as follows:

- We develop a partitioning routine to divide the workflow into several distinct partitions. In this routine, at each partition, tasks can be processed independently of each other and simultaneously on different machines. We have used this simple and efficient routine to produce a variety of chromosomes, which results in faster convergence to the optimal solution as the population is not randomly generated.

Table 1 Notations used in DCHG-TS

Notation	Definition
P_i	The th processor in the system
T_i	The th task in the DAG of workflow
T_{entry}	The entry task with no predecessor in the workflow
T_{exit}	The exit task with no successor in the workflow
DAG	The directed acyclic graph
$w(t_i)$	The task weight, representing the execution time of t_i
(ct_i,t_j)	The cost of data transferring between t_i and t_j
$SUCC(t_i)$	The set successor tasks after t_i
LT	The level of task
N_P	The size of the population
λ_1	The deadline factor for the multi-objective fitness function
λ_2	The cost factor for the multi-objective fitness function
P_m	The probability of mutation
S_{length}	The scheduling length for the corresponding chromosome
ECT	The expected completion time
ECC	The expected completion cost

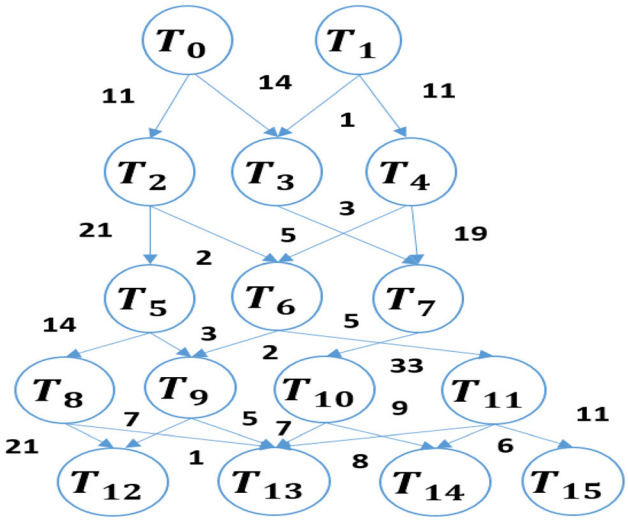


Fig. 2 A simple DAG containing 15 tasks

- We use the solution of Heterogeneous Earliest Finish Time (HEFT) heuristic in the initial population of Deadline-aware and Cost-effective Hybrid Genetic Task Scheduling (DCHG-TS). This efficient and robust heuristic has low time complexity and makes the initial population of DCHG-TS more efficient, and thus the algorithm converges efficiently in less time.
- We develop new and modified operators (inversion, improved crossover, and mutation) to enable DCHG-TS to explore large problem space in less time and to provide efficient solutions.
- Another advantage that our proposed algorithm has over other task scheduling algorithms is at execution time; a load balancing routine is used to improve the efficiency of the resources, so the resources are not overloaded or stayed idle for a long time. This routine distributes the load equally across all resources, which results in efficient scheduling and less makespan, especially in large cloud systems.

4 The proposed algorithm

4.1 Chromosome representation

The population consists of several individuals called chromosomes. Each of the chromosomes consists of two parts. The first gene in each chromosome represents the host processor, and the following genes will determine the order of scheduled tasks on that processor. The priority constraints among tasks should not be violated; otherwise, the chromosome will become an invalid chromosome. Each chromosome represents the order of the tasks that are scheduled on a processor. An example of the chromosome used in this paper is shown in Fig. 3.

4.2 Initial population

In CWTS-GA, except for one chromosome, the rest of the chromosomes are randomly generated, which leads to search large problem spaces inefficiently. Thus, in DCHG-TS, the initial population is generated, so optimal solutions

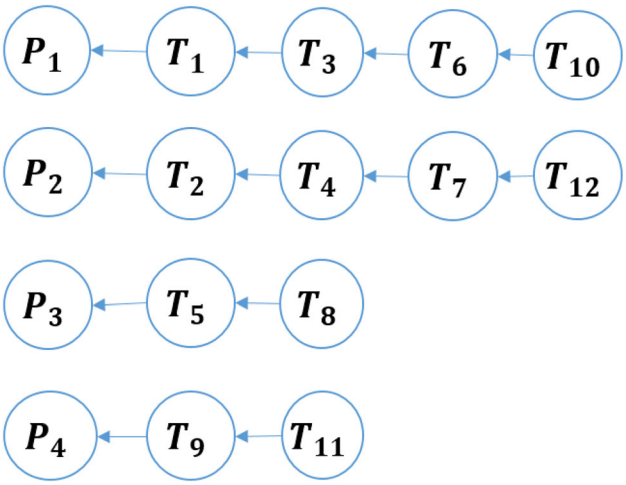


Fig. 3 An example of the chromosome

are obtained at an acceptable time.

Algorithm 1: initial population routine

Input: a DAG of workflow, NP population size
Output: the initial population
1: for i=1 to N_p-1 do
2: foreach partition in DAG
3: foreach task in level
4: calculate LT for each task;
5: sort tasks based on descending order of LT and insert them in T-queue;
6: pickup task from T-queue and assign it to the fastest available processor;
7: endfor
8: endfor
9: endfor

Algorithm 1 uses the partition function to divide the workflow into several independent partitions (Fig. 7). Each partition should be such that it covers the independent tasks that can be performed on different processors. This operation is used to assign tasks to different processors to create high diversity in chromosomes. In steps 1 to 3, each task of DAG of the workflow selects. Then calculates the value of LT using Eq. (2) for each task (Step 4) and in a task queue (T-queue), the tasks are placed by considering LT in descending order (step 5). A task with a higher LT value exists in a critical path for processing, so from the beginning of the T-queue, each of the tasks is removed and assigned to the available fastest processor (Step 6). Finally, the initial efficient population is generated.

4.3 Fitness calculation

The fitness function plays an essential role in the genetic algorithm. If the defined constraints are stringent, then the purpose of this function becomes more perceptible. In this paper, the fitness function after calculating the penalty is defined as Eq. (3).

$$\text{Fitness}\ (\mathrm{X_i}) = \mathrm{C/M(X_i)} \tag{3}$$

Where $\mathrm{M(X_i)}$ is defined as Eq. (4).

$$\mathrm{M(X_i)} = \lambda_1 \frac{T(X_i)}{dead\_time} + \lambda_2 \frac{C(X_i)}{Max\_cost} \tag{4}$$

Where is the time of completion of the scheduling scheme provided by the chromosome X_i, is the user-defined deadline, $C(X_i)$ represents the total cost of fully implementing the scheduling provided by the chromosome , Max_cost is the maximum cost defined by the user, λ_1 and λ_2 denote the user-defined QoS parameters, which specifies the deadline and cost priority, respectively,$0 \le \lambda_1 \le 1$ and $0 \le \lambda_2 \le 1$ are such that $\lambda_1 + \lambda_2 = 1$. $\lambda_1 - \lambda_2 = 0$ represents that both QOS parameters for the user are the same preference, $\lambda_1 - \lambda_2 > 0$ indicates that the deadline parameter is more critical to the user, $\lambda_1 - \lambda_2 < 0$ indicates that the fitness function emphasizes choosing less costly chromosomes.

The evaluation process is such that if $\frac{T(X_i)}{dead\_time} > 1$ or $\frac{C(X_i)}{Max\_cost} > 1$, then this chromosome is infeasible and should be removed from the population; otherwise, the chromosome is feasible and maintained.

4.4 Crossover operator

The selection of the elite chromosomes in this paper is performed through the roulette wheel method [10][28, 29], which is an ideal option to implement the selection operation in the genetic algorithm and then, the multi-fold and modified crossover operator have been used for the selected chromosomes. The crossover operator used in this paper is nearly twice as efficient as other existing crossover operators used in task scheduling algorithms. In single-point crossover mode, a point of the chromosome selected randomly, but in double-point crossover mode, two points of the chromosome are chosen randomly. Meanwhile, in triple crossover mode, three points of the chromosome are selected randomly.

The details of the crossover operation are shown in Fig. 4. Two father's and mother's chromosomes are shown in Fig. 4a, and after the crossover operator is performed, in Fig. 4b, two new chromosomes are produced son = {{(p_1, t_1, t_8, t_7, t_{10}, t_{15}), (p_2, t_2, t_3, t_{12}, t_9, t_{11}), (p_3, t_4, t_5, t_6, t_{13}, t_{14})}} and daughter = {{(p_1, t_2, t_5, t_6, t_9, t_{11}), (p_2, t_4, t_3, t_7, t_{13}, t_{14}), (p_3, t_1, t_8, t_{12}, t_{10}, t_{15})}}.

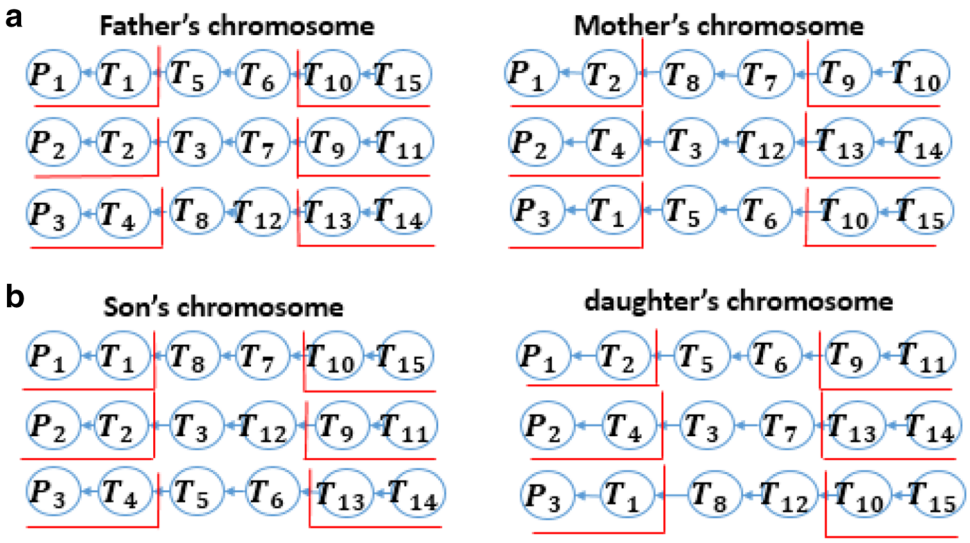


Fig. 4 **a** Example of parent chromosomes. **b** Example of two-point crossover

The details of the crossover operation are given in Algorithm 2.

Algorithm 2: Crossover routine

Input: two points from the current population
Output: two new chromosomes
Data: Number of crossover ($N_c = N_s/2$)
1: for k=1 to N_c
2: Randomly select a father's chromosome and mother's chromosome from the current population;
3: Generate two chromosomes X_c and X_d by single-point crossover;
4: Generate two chromosomes X_e and X_f by double-point crossover;
5: Generate two chromosomes X_g and X_h by triple-point crossover;
6: Calculate the fitness values of X_c, X_d, X_e, X_f, X_g, and X_h;
7: Select two chromosomes with the highest fitness values as N_s from the chromosomes calculated in step 6;
8: end for

Algorithm 2 presents the advanced crossover routine compared to the existing genetic algorithms. In step 1, the number of repetitions of the operation is equal to (N_c =) (where is the selected population). In step 2, two random chromosomes are selected from the population as parental chromosomes (X_a and X_b). X_c and X_d chromosomes are produced by the single-point crossover operator in step 3. In particular, the other two chromosomes X_e and X_f are produced by the double-point crossover operator on the same parent chromosomes (X_a and X_b) in step 4. Two chromosomes X_g and X_h, are obtained from the triple-point crossover operator on parent chromosomes (X_a and X_b) in step 5. In step 6, using Eq. (3) the fitness of these chromosomes is evaluated. In step 7, the best two chromosomes with the highest fitness values are selected and added to the population (N_S).

4.5 Mutation operator

In the proposed algorithm, two modes have been used in the mutation operator, including the inner mutation and the outer mutation. It should be noted that in both operating modes, the combination of single-point mutation and double-point mutation is used, which, unlike other genetic algorithms introduced in this field, increases the algorithm's efficiency. In single-point mode, two tasks are selected randomly, and the order of their execution is exchanged within a processor. Then the fitness values of the obtained chromosomes are compared to each other, and the chromosome with a higher amount of fitness is selected.

In mutation operation with double-point mode, two tasks are selected randomly and assigned to two selected processors randomly, such that these processors have the capabilities to process these tasks. Choosing external or internal operations is random, and according to the probability of mutation p_m, a random number is generated between 0 and 1. If this number is smaller than p_m, the inner operation is selected, and otherwise, the outer operation is selected for the mutation operator.

The operation of the inner mutation and the outer mutation is shown in Figs. 5, and 6, respectively.

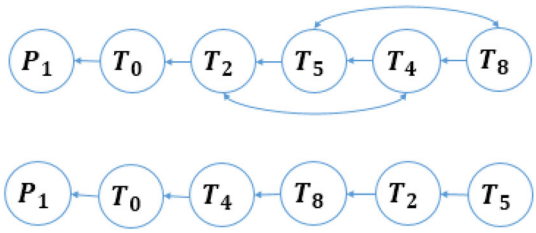


Fig. 5 Example of inner mutation with double-point mode

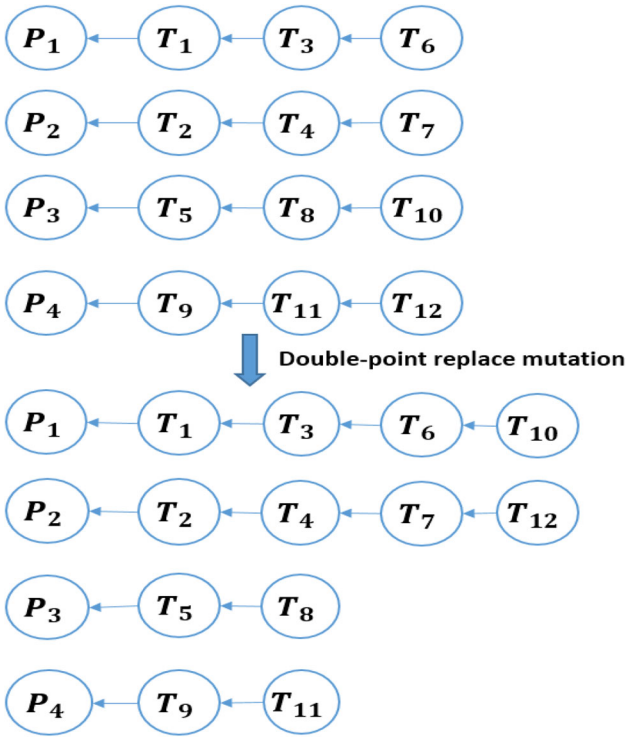


Fig. 6 Example of outer mutation with double-point mode

Algorithm 3: Mutation routine

Input: one chromosome from the current population
Output: new generated chromosome
Data: number of mutations n = $P_m \times N_s$

1: for i=1 to n do
2: randomly generate a number R between 0 and 1;
3: if (R < P_m) {
4: randomly select a chromosome X_i from the current population;
5: perform inner mutation with single-point mode offspring = X_j;
6: perform inner mutation with double-point mode offspring = X_k;
7: compute fitness of X_j and X_k;
8: select offspring with better fitness value as N_s;}
9: else {
10: randomly select a chromosome X_l from the current population;
11: perform outer mutation with single-point mode offspring = X_m;
12: perform outer mutation with double-point mode offspring = X_n;
13: compute fitness of X_m and X_n;
14: select offspring with better fitness as N_s;}
15: end for

The presence of two inner and outer operators for mutation operation has led to an increase in algorithm efficiency. In step 1 of algorithm 3, the algorithm is repeated for the number of mutations (n = $P_m \times N_s$) where P_m is the probability of mutation. In step 2, the number R is randomly generated between 0 and 1. If ($R < P_m$), then both the single-point and double-point inner mutations are performed, and then the best chromosomes are selected using Eq. (3) and added to the population (steps 4 to 8). Else if ($P_m < R$), then outer mutation operations will be performed as both single-point and double-point procedures (Fig. 6), and finally, the superior chromosomes will be added to the population in terms of their fitness (Steps 10 to 14).

4.6 Inversion operator

The proposed algorithm uses a partition routine such that the tasks of each partition can be executed independently. An example of a workflow application and the corresponding partition is shown in Fig. 7.

An additional function of DCHG-TS compared to the recent task scheduler based on genetic algorithms is an inversion operator. The two-mode inversion operator has been used in this paper that consists of the exchange and the replace mode. This operator exchanges the order of tasks of the same partition, in which these tasks can be executed independently. The details of this operator are shown in Fig. 8.

4.7 Algorithm termination

The algorithm is executed until the termination condition is satisfied. The number of generations in the proposed algorithm is taken as 100.

4.8 Load balancing operator

DCHG-TS optimizes the load balancing during the execution time such that processors completed the task efficiently, did not stay idle or overloaded for a long time.

$$\text{Min\_Pt} = \text{minimum}\ \{pt_1, \ldots, pt_n\} \tag{5}$$

$$\text{Load\_w} = S_{\text{length}} - \text{Min\_Pt} \tag{6}$$

The load balancing routine of the proposed algorithm predicts the available resources and calculates the difference between the execution time of each chromosome and the lowest finish time of each processor. Eq. (5) calculates minimum processing time for processors. In Eq. (5), pt_i is the processing finish time of the processor i. Eq. (6) calculates the value of Load_w using the difference between S_{length} and Min_Pt, Where Min_Pt is minimum finish time

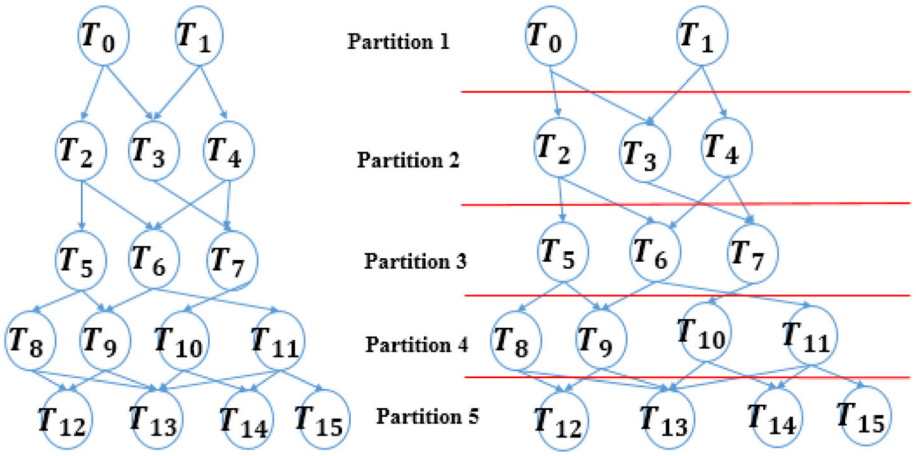


Fig. 7 An example of workflow DAG and corresponding partitions

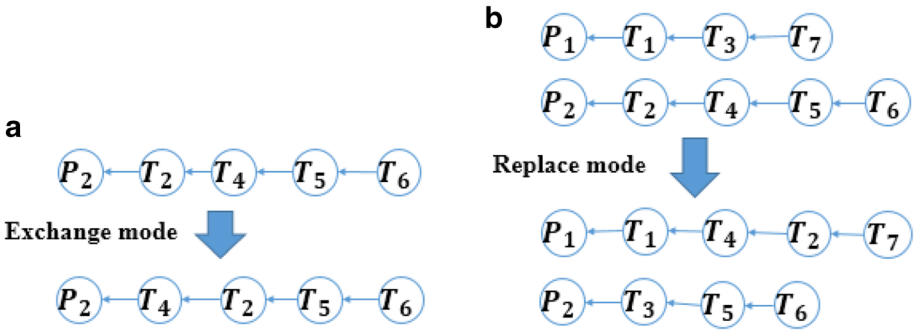


Fig. 8 Inversion operations: **a** Exchange **b** Replace

of processors, and S_{length} is The scheduling length for the corresponding chromosome. The high values for Load_w are not optimal; hence, half of the chromosomes with higher values of Load_w are selected and put in a queue according to their value of Load_w in not decreasing order. The worst chromosomes are picked up from the queue, and the overloaded processor is identified, then the task of the overloaded processor is assigned to another random processor that is capable of processing this task. In the end, the fitness of the new chromosome is calculated. If the new chromosome's fitness is better than the previous chromosome, then the new chromosome is replaced by the earlier chromosome.

4.9 DCHG-TS algorithm flow

Step 1: Determine the initial values for the Crossover and Mutation probability, the Elitism rate, the Population size, the number of processors, the execution time matrix ECT, and the execution cost matrix ECC.

Step 2: Call HEFT algorithm and seed HEFT solution as a chromosome in the initial population.

Step 3: Call the partitioning routine and assign tasks on the critical path of the DAG to the fastest processors to generate the remaining chromosomes of the initial population.

Algorithm 4: load balancing routine

```
1: calculate the amount of load_w for all the schedulers;
2: select the chromosomes whose load_w values have the most difference with the load_w threshold;
3: for the selected chromosomes, do the following steps
4:     identify the processor that has been overloaded and assign its tasks to a processor that is capable of performing those tasks;
5:     if (fitness (new chromosome) > fitness (current chromosome))

6:         replace new chromosome
7:     end if
8: end for
```

Step 4: Until the termination condition is not satisfied, the following steps should be executed unless the termination condition is met, then the operation stops.

Step 5: compute the fitness value of each chromosome in the evaluation stage.

Step 6: Considering S = Elitism rate × Population size, select the chromosomes with the highest fitness values.

Step 7: Call the multi-fold crossover routine, create new chromosomes, and add to the selected population.

Step 8: The number of mutation operations is equal to the mutation rate × The number of selected chromosomes, and by performing mutation operations on the selected chromosomes, new chromosomes will be created and added to the selected population.

Step 9: For all chromosomes, for each partition in each chromosome, task inversion operation is done.

Step 10: Call load balancing routine.

Step 11: Return step 5 until genetic operation stops.

5 Simulation and performance analysis

The efficiency of DCHG-TS has been compared to several the state of the art algorithms by simulation experiments in the CloudSim environment. Both real-world workflows and synthesized workflows have been used to evaluate the proposed algorithm.

5.1 Experimental settings

After the number of simulations, the most suitable parameters for the proposed algorithm that provide the best results with the mutation and crossover rates of 0.01 and 0.8, respectively. The population size and the number of generations for random workflows are taken as 100 to simplify the simulations. The chosen metric for performance evaluation is Normalized Scheduling Cost (NSC), which is achieved by dividing the total cost of the scheduling and the cost of executing a workflow on the cheapest processor. Another parameter used to evaluate performance is the Communication to Computation Rate (CCR) [27]. The state of the art algorithms GAACO, PGA, and CWTS-GA are used to evaluate the efficiency ofDCHG-TS.

The data transfer times and information of the task processing times are presented in Fig. 2; Table 2, respectively, when the processors operate at their maximum frequency. In experiments, pricing calculations are based on the study of Zheng [30].

Our proposed algorithms and experimental simulations are implemented by java (JDK 8). The settings for generating 100 random DAGs and the input parameters for proposed task scheduling algorithms are shown in Table 3. As the computation of workflows increases, the advantages of DCHG-TS become more apparent. Due to scheduling algorithms should be highly efficient against complex and large workflows with high complexity. Therefore, in experiments, workflows with a large number of nodes also have been used as inputs, as shown in Table 4. In Fig. 9, $\lambda_1$is the deadline factor, and the horizontal axis shows the settings for λ_1, and since $\lambda_1 + \lambda_2 = 1$, so λ_2 will also be set.

Table 2 An example of task execution times with the highest CPU frequency

Tasks	Processor 1	Processor 2	Processor 3
Task 1	60	24	32
Task 2	54	42	144
Task 3	8	72	12
Task 4	42	12	40
Task 5	40	32	162
Task 6	56	12	96
Task 7	70	28	14
Task 8	10	56	60
Task 9	42	6	72
Task 10	56	4	128
Task 11	30	12	16
Task 12	27	21	72
Task 13	4	36	6
Task 14	21	6	20
Task 15	20	16	81

5.2 Performance evaluation

In this section, the performance of DCHG-TS is evaluated. As can be seen in Fig. 9, as factor λ_1 increases, the normalized scheduling cost for all three methods decreases. Compared to the other two methods, DCHG-TS has a lower reduction rate of NSC due to the other two methods do not consider the cost factor in the schedule and also, by increasing the amount of λ_1, DCHG-TS gives higher priority to the cost.

Sets of workflow 5 k, 10 k, and 15 k are generated randomly with different CCR, and the simulation experiments results are obtained with distinct CCR. As such, one of the performance parameters we have used to evaluate the efficiency of DCHG-TS compared to other algorithms in experiments is Average Schedule Length (ASL), which is obtained by 1000 runs of the algorithm. The results are

Table 3 Random DAG production parameters for input different workflow scheduling algorithms

DAG	Type	Number of DAGs	CCR	Number of processors	Mutation rate	Crossover rate	Population size	Number of generations
	Random	100	0.1,0.5,1.0,2.0, 5.0,10	3	[0.01 1]	[0.01 1]	100	100

Table 4 Inspiral DAG production parameters for evaluating workflow scheduling algorithms

DAG	Type	Number of DAGs	CCR	Number of processors	Mutation rate	Crossover rate	Population size	Number of generations
	Inspiral	1000	1	[2 35]	[0.01 1]	[0.01 1]	1000	1000

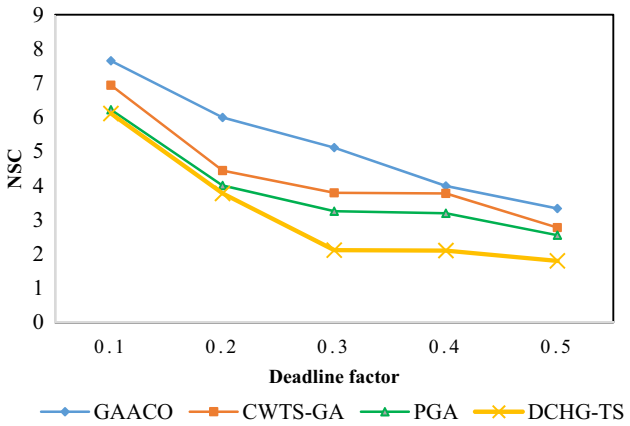


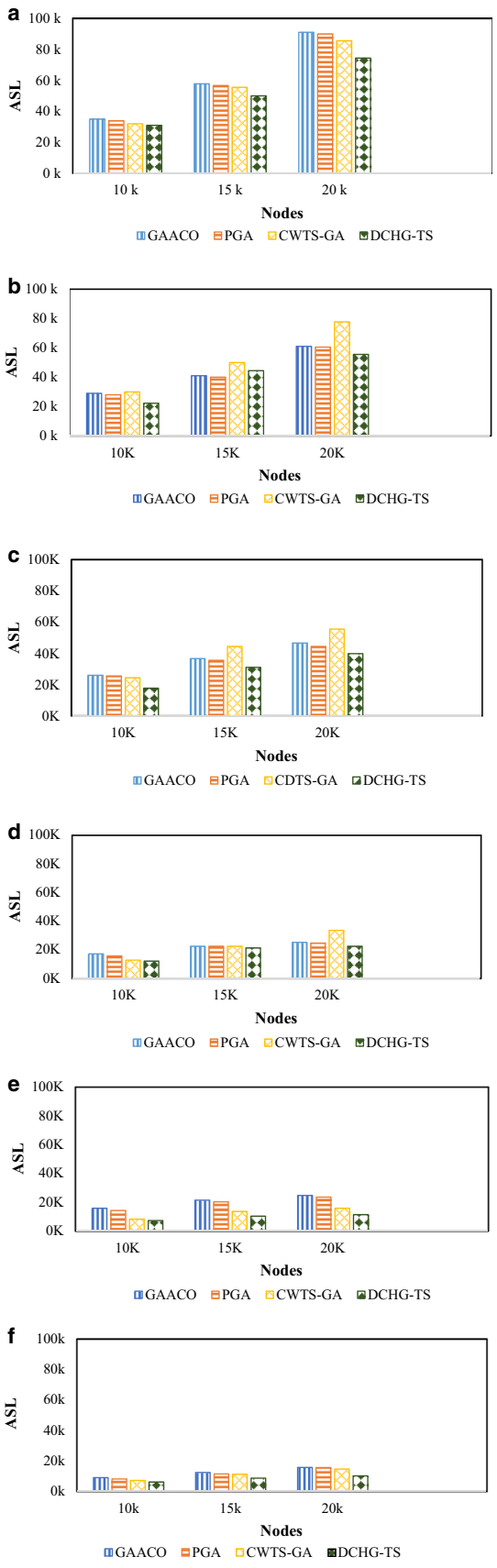
Fig. 9 Normalized scheduling cost with increasing deadline factor

shown in Fig. 10. DCHG-TS has a significant advantage over GAACO, PGA, and CWTS-GA for the CCR values of 0.1, 1, and 10. Various random workflows with computational and communication characteristics have been investigated to analyze the efficiency of DCHG-TS from the ASL perspective. In terms of ASL, DCHG-TS is better than GAACO, PGA, and CWTS-GA by almost 25.77%,19.69%, and 13.58%, respectively. Figure 10 shows that DCHG-TS continues to perform better than other algorithms with the increasing number of workflow nodes. GAACO has more unsatisfactory results than other algorithms because it uses a random initial population, while DCHG-TS uses an efficient initial population. As is evident, as the number of nodes increases, the ASL rate also increases for all algorithms.

Also, in this paper, the real-world workflow LIGO's Inspiral [31] is used to evaluate the performance of the proposed algorithm. The Laser Interferometer Gravitational-wave Observatory (LIGO) is trying to detect gravitational waves generated by various events in the universe, according to Einstein's general ratio. LIGO's inspiral workflow is used to analyze data from the correlation of compact binary systems such as binary neutron stars and black holes. The time-frequency data is divided into smaller blocks by all LIGO detectors for analysis. For each block, the workflow generates a subset of the waveforms that belong to the parameter space and calculates the output of the matched filter. If a real Inspiral is discovered, a trigger will be created that can be checked with the triggers of other detectors.

Meanwhile, several additional compatibility tests may also be added to the workflow. The Inspiral workflow is data-driven and presented as a DAX XML format. Here, we introduce the Inspiral workflow with 1000 nodes as the real-world input workflow to evaluate the performance of DCHG-TS compared to other algorithms.

In Fig. 11, the performance comparison of the proposed algorithm with the other three algorithms for increasing the number of processors can be seen. The number of processors is considered to be from 2 to 10, so that one unit is increased in each experiment. The GAACO, PGA, DCHG-TS, and CWTS-GA algorithms are implemented with 1000-node Inspiral workflow according to the number of processors allocated. The performance parameter measured in these experiments is makespan, and the number of processors is an adjustable parameter. As shown in Fig. 11, the algorithms' efficiency improves with an increasing number of processors due to increased processing parallelism. However, due to the serial nature of the Inspiral workflow, as well as the amount of overhead is imposed on the system by increasing the number of processors, efficiency improvement is limited to a certain number of processors. The superiority of our proposed algorithm is using the efficient initial population, and also the load balancing routine to increase the efficiency of the processors in the runtime. However, in other algorithms, the



◀**Fig. 10** Average Scheduling Length of synthesized workflows for CCR **a** 0.1, **b** 0.5, **c** 1.0, **d** 2.0, **e** 5.0, **f** 10.0

initial population is generated randomly, and moreover, an efficient load balancing routine is not considered.

In Fig. 11, when the minimum number of processors (e.g., the number of processors is 2), in terms of average performance, our proposed algorithm is outperformed CWTS-GA, PGA, and GAACO by almost 4%, 5%, and 6%, respectively. However, as the number of processors increases, the rate of efficiency increases dramatically. When the number of processors is equal to 6, experimental results show that DCHG-TS effectively improves average performance by at least 13%, 17%, and 28% compared to CWTS-GA, PGA, and GAACO, respectively. In the case of the maximum number of processors, when the number of processors is equal to 10, DCHG-TS improves the performance by 30%,33%, and 48% compared with CWTS-GA, PGA, and GAACO, respectively.

5.3 Time complexity and computation overhead evaluation

Finally, we evaluate the computational overhead of each algorithm. For each algorithm, an average of 1000 executions was implemented with the Inspiral workflow with 1000 nodes. All algorithms are performed on a server with 8-core Intel® Core™ i7-9750H 2.6-GHz CPU and 16 GB of DDR4 RAM. The result of the algorithm execution time is measured by increasing the number of resources.

As shown in Fig. 12, the computational overhead of DCHG-TS is much lower than the other three algorithms, regardless of the number of processors. As can be seen, as the number of processors increases, more time overhead is imposed on the algorithms. Besides, when the number of processors is 25, DCHG-TS and CWTS-GA time overhead is less than 60 s. Compared to the makespan for the Inspiral workflow with 1000 nodes, such a time overhead makes sense for DCHG-TS. Specifically, when the number of processors exceeds 30, DCHG-TS is more acceptable than the other three algorithms.

6 Discussions

From the simulation experiments results obtained and comparing the three state of the art algorithms to DCHG-TS, it is evident that DCHG-TS has the following advantages:

1 the fitness function of the proposed algorithm has considered both deadline and budget constraints of the

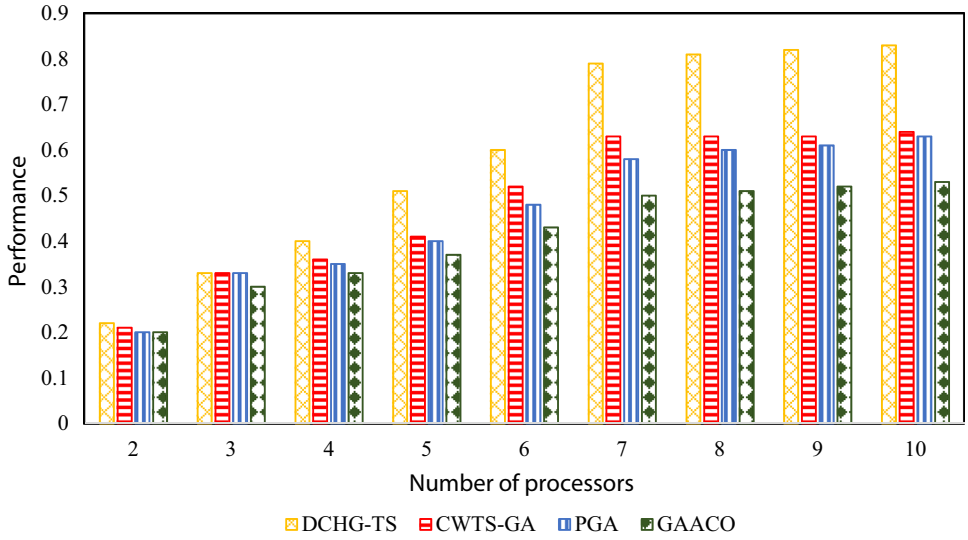


Fig. 11 Performance comparison for Inspiral workflow with 1000 nodes

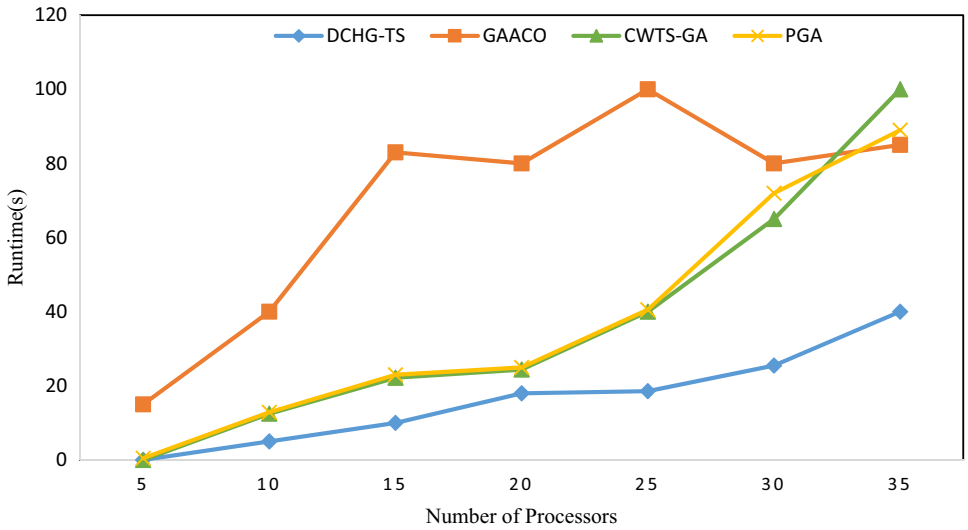


Fig. 12 Comparison of runtime algorithm results

cloud workflow scheduling problems. Most of the proposed algorithms rely solely on deadline optimization and do not focus on cost reduction; meanwhile, the proposed algorithm takes into account both the quality of service parameters.

2 DCHG-TS algorithm uses multi-fold crossover and mutation operators. Moreover, new routines such as inversion operator, have generated more diverse and logical populations, which increases the performance of the proposed algorithm compared to the state of the art algorithms.

3 The chromosomes are encoded as two-dimensional forms. Moreover, instead of using a random initial population, the directed population is generated by using the solution obtained of HEFT heuristic and the efficient initial-population routine.

4 DCHG-TS uses the benefits of an efficient load-distribution routine during execution. At the same time, most of the state of the art and new algorithms in this field lacked such routine, which caused a decrease in the processor's efficiency.

7 Conclusion

In this paper, a modified and hybrid genetic algorithm is presented. This algorithm uses a robust heuristic for workflow scheduling on clouds. In the proposed algorithm to obtain the optimal solution with the least number of iterations, the initial population has been replaced with a population with optimal solutions, and the HEFT heuristic schedule as one of the chromosomes of the initial population has been seeded. The workflow is partitioned, such

that the tasks of each partition are independent of each other and can be processed simultaneously. Each chromosome is encoded in two-dimensional form. In the proposed algorithm, the rigorous search is performed using multi-fold crossover and mutation operators, which covers the vast and complex problem space and increases the proposed algorithm's performance. In DCHG-TS, a load balancing routine is also used to ensure the optimal performance of resources. Simulation Experiments with different datasets of different sizes proved the scalability and diversity of DCHG-TS. Comparison results with the state of the art algorithms Indicate that DCHG-TS is better than other algorithms in terms of scheduling quality because DCHG-TS reduces both the cost and the length of the scheduling. In the future, we plan to consider other parameters involved in reducing bottleneck and increasing the overall efficiency of the system, including task data locality and running task preemption. Also, we intend to use a combination of different efficient heuristics in DCHG-TS.

References

1. Atkinson, M., et al.: The DATA Bonanza: Improving Knowledge Discovery in Science, Engineering, and Business. Wiley, Hoboken (2013)
2. Bokhari, M.U., Makki, Q., Tamandani, Y.K.: A Survey on Cloud Computing, pp. 149–164. Springer, Singapore (2018)
3. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Futur. Gener. Comput. Syst. **25**(6), 599–616 (2009)
4. Wu, F., Wu, Q., Tan, Y.: Workflow scheduling in cloud: a survey. J. Supercomput. **71**(9), 3373–3418 (2015)
5. Rodriguez, M.A., Buyya, R.: Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. IEEE Trans. Cloud Comput. **2**(2), 222–235 (2014)
6. Yuan, H., Liu, H., Bi, J., Zhou, M.: Revenue and energy cost-optimized biobjective task scheduling for Green Cloud Data Centers. IEEE Trans. Autom. Sci. Eng. (2020). https://doi.org/10.1109/TASE.2020.2971512
7. Cui, Y., Xiaoqing, Z.: Workflow tasks scheduling optimization based on genetic algorithm in clouds. In: 2018 3rd IEEE International Conference on Cloud Computing and Big Data Analysis, pp. 6–10. ICCCBDA, Chengdu (2018)
8. Liu, L., Zhang, M., Buyya, R., Fan, Q.: Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing. Concurr. Comput. Pract. Exp. **29**(5), e3942 (2017)
9. Wang, X., Yeo, C.S., Buyya, R., Su, J.: Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. Futur. Gener. Comput. Syst. **27**(8), 1124–1134 (2011)
10. Xu, Y., Li, K., Hu, J., Li, K.: A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Inf. Sci. (Ny) **270**, 255–287 (2014)
11. Li, H., Wang, L., Liu, J.: Task scheduling of computational grid based on particle Swarm Algorithm. In: 2010 Third International Joint Conference on Computational Science and Optimization, pp. 332–336. IEEE, Piscataway (2010)
12. Basu, S., et al.: An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment. Futur. Gener. Comput. Syst. **88**, 254–261 (2018)
13. Kimpan, W., Kruekaew, B.: Heuristic task scheduling with artificial bee colony algorithm for virtual machines. In: Proceedings – 2016 Joint 8th International Conference on Soft Computing and Intelligent Systems and: 2016 17th International Symposium on Advanced Intelligent Systems, SCIS-ISIS 2016, pp. 281–286. Piscataway, IEEE (2016)
14. Wang, J., Li, X., Ruiz, R., Yang, J., Chu, D.: Energy Utilization Task Scheduling for MapReduce in Heterogeneous Clusters, IEEE Transactions on Services Computing. Piscataway, IEEE (2020)
15. Sun, H., Yu, H., Fan, G.: Contract-Based Resource Sharing for Time Effective Task Scheduling in Fog-Cloud Environment. IEEE Transactions on Network and Service Management. IEEE, Piscataway (2020)
16. Tang, Z., Qi, L., Cheng, Z., Li, K., Khan, S.U., Li, K.: An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. J. Grid Comput. **14**(1), 55–74 (2016)
17. Yadav, R., Zhang, W., Li, K., Liu, C., Shafiq, M., Karn, N.K.: An adaptive heuristic for managing energy consumption and overloaded hosts in a cloud data center. Wirel. Networks **26**(3), 1905–1919 (2020)
18. "MeReg: Managing Energy-SLA Tradeoff for Green Mobile Cloud Computing." [Online]. Available: https://www.hindawi.com/journals/wcmc/2017/6741972/. Accessed: 21-Apr 2020
19. Li, H., Zhu, G., Cui, C., Tang, H., Dou, Y., He, C.: Energy-efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing. Computing **98**(3), 303–317 (2016)
20. Garg, R., Mittal, M., Son, L.H.: Reliability and energy efficient workflow scheduling in cloud environment. Cluster Comput. **22**(4), 1283–1297 (2019)
21. Arabnejad, V., Bubendorfer, K.: Cost effective and deadline constrained scientific workflow scheduling for commercial clouds. In: Proceedings - 2015 IEEE 14th International Symposium on Network Computing and Applications, NCA 2015, pp. 106–113. Piscataway, IEEE (2016)
22. Keshanchi, B., Souri, A., Navimipour, N.J.: An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing. J. Syst. Softw. **124**, 1–21 (2017)
23. Ghobaei-Arani, M., Souri, A., Safara, F., Norouzi, M.: An efficient task scheduling approach using moth-flame optimization algorithm for cyber-physical system applications in fog computing. Trans. Emerg. Telecommun. Technol. **31**(2), e3770 (2020)
24. Mortazavi-Dehkordi, M., Zamanifar, K.: Efficient deadline-aware scheduling for the analysis of Big Data streams in public Cloud. Cluster Comput. **23**(1), 241–263 (2020)
25. Kaur, G., Kalra, M.: Deadline constrained scheduling of scientific workflows on cloud using hybrid genetic algorithm. In: *Proceedings of the 7th International Conference Confluence 2017 on Cloud Computing, Data Science and Engineering, pp. 276–280. IEEE, Piscataway (2017)*
26. Lam, A.Y.S., Li, V.O.K.: Chemical-Reaction-Inspired Metaheuristic for Optimization. IEEE Trans. Evol. Comput. **14**(3), 381–399 (2010)
27. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)

28. Kumar, N., Vidyarthi, D.P.: A novel hybrid PSO–GA meta-heuristic for scheduling of DAG with communication on multi-processor systems. Eng. Comput. **32**(1), 35–47 (2016)
29. Ahmad, S.G., Liew, C.S., Munir, E.U., Ang, T.F., Khan, S.U.: A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems. J. Parallel Distrib. Comput. **87**, 80–90 (2016)
30. Zheng, W., Qin, Y., Bugingo, E., Zhang, D., Chen, J.: Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds. Futur. Gener. Comput. Syst. **82**, 244–255 (2018)
31. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Futur. Gener. Comput. Syst. **29**(3), 682–692 (2013)

Amir Iranmanesh graduated in computer engineering from shahid bahonar university, kerman, iran, in 2007 and received his master's degree in information technology engineering from graduate university of advanced technology, iran, in 2017. His research interests include task scheduling, evolution algorithm, data-intensive workflow optimization and big data analytics.

Hamid Reza Naji is Experienced Associate Professor with a demonstrated history of working in the higher education and industry. Skilled in IT and Computer Science, Embedded Systems & Field-Programmable Gate Arrays (FPGA), Network and Security. Strong education professional focused in Computer Engineering with a Bachelor of Science (BS) from Shiraz University, Master of Science (M.Sc.) from Tehran University and PhD from University of Alabama at Huntsville, AL, USA. He is currently an assistant professor in the department of computer science at the Graduate University of Advanced Technology, Iran.