Neural Computing and Applications (2020) 32:15263–15278
https://doi.org/10.1007/s00521-020-04878-8

ORIGINAL ARTICLE

A hybrid genetic algorithm for scientific workflow scheduling in cloud environment

Hatem Aziza[1] · Saoussen Krichen[1]

Received: 23 March 2019 / Accepted: 20 March 2020 / Published online: 11 May 2020

Abstract
Nowadays, we live an unprecedented evolution in cloud computing technology that coincides with the development of the vast amount of complex interdependent data which make up the scientific workflows. All these circumstances developments have made the issue of workflow scheduling very important and of absolute priority to all overlapping parties as the provider and customer. For that, work must be focused on finding the best strategy for allocating workflow tasks to available computing resources. In this paper, we consider the scientific workflow scheduling in cloud computing. The main role of our model is to optimize the time needed to run a set of interdependent tasks in cloud and in turn reduces the computational cost while meeting deadline and budget. To this end, we offer a hybrid approach based on genetic algorithm for modelling and optimizing a workflow-scheduling problem in cloud computing. The heterogeneous earliest finish time (HEFT), an heuristic model, intervenes in the generation of the initial population. Based on results obtained from our simulations using real-world scientific workflow datasets, we demonstrate that the proposed approach outperforms existing HEFT and other strategies examined in this paper. In other words, experiments show high efficiency of our proposed approach, which makes it potentially applicable for cloud workflow scheduling. For this, we develop a GA-based module that was integrated to the WorkflowSim framework based on CloudSim.

Keywords Cloud computing · Genetic algorithm · Scientific workflow · Workflow scheduling · Deadline · Budget

1 Introduction

Over last few years, cloud computing (CC) has become an emerging research area. It is considered as the main model of distributed computing. It offers elastically scalable and highly available resources as a subscription-based service like utility computing [1] for executing scientific workflows (SWfs). The SWf (such as Montage, CyberShake, Epigenomics, LIGO and SIPHT) is a transposition of the general term of workflow in the experimental context, that is to say relating only to computational processes via complex data flows and control dependencies while automating their implementations in appropriate resources [2].

Successful execution of SWf requires optimal use of resources. For that, the work must be focused on finding the best strategy for allocating workflow tasks to available computing resources. This is called workflow scheduling (WS). The WS aims at mapping and managing the execution of interdependent tasks by considering precedence constraints on shared resources [3]. This problem is known to be $\mathcal{NP}$-Complete [3] due to its combinatorial aspect. What prompted researchers to provide a near-optimal solution.

Workflows must be well defined and managed to be executed later. This is the role of an efficient workflow management system (WMS).

As shown in Fig. 1, to schedule and map workflow tasks to the available resources in cloud environment, system needs a workflow scheduler (bridge).

✉ Hatem Aziza
hatem.aziza@isg.rnu.tn

Saoussen Krichen
saoussen.krichen@isg.rnu.tn

[1] Institut Supérieur de Gestion, LARODEC, Université de Tunis, Tunis, Tunisia

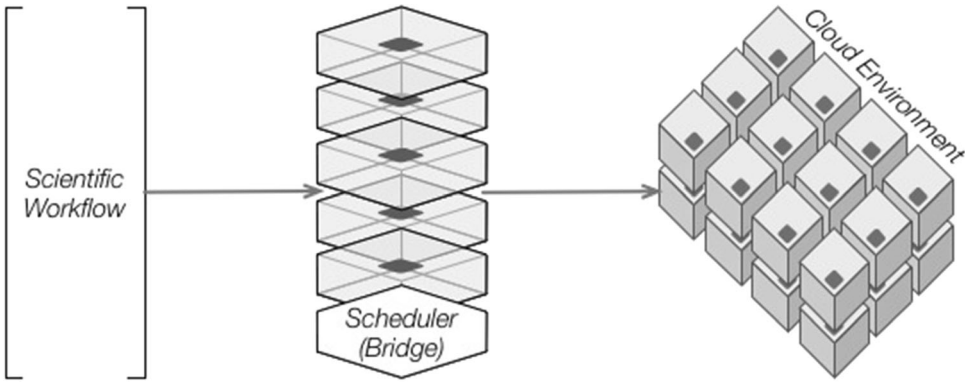

Fig. 1 Scientific workflow execution architecture in cloud environment

In this work, we extend the solution proposed in our previous work [4] by dealing with the SWf scheduling in cloud environment. We focus on efficient use of resources in order to run a SWf composed of interdependent tasks.

To this end, we develop a new hybrid GA for SWf scheduling in CC. The main idea of the proposed solution is matching a SWf tasks to proper resources in order to minimize the computation cost and the execution time while meeting deadline and budget.

The tasks scheduling in CC is among the most important problems for the various stakeholders in this environment especially when it comes to the scheduling of SWf at different levels of difficulty and complexity.

In the scientific field, we can find workflows sensitive to large volumes of data, others sensitive to complex processing and those sensitive to different criteria at the same time.

This very important subject has prompted several researchers to propose solutions aimed at optimizing the processing of these SWfs and in particular seeking a compromise between two contradictory quality of service (QoS) parameters which are time and cost. In this context, it should be noted that QoS determines the level of satisfaction of a user of a given service. Generally, the quality of service is measured by qualitative metrics such as computational time, computational cost and reliability. Generally, to process workflows very quickly, you have to hire powerful resources that are costly. On the other hand, cheaper resources can be slow to complete a job.

This contradiction pushed us to set up a promising solution aiming to minimize the processing time while minimizing the cost of this processing as much as possible and respecting the deadline and budget constraints.

To achieve this dual objective, we have looked in the literature for the most used approaches that can meet this objective by giving good results. For this, we have chosen to mix the power and simplicity of HEFT with the evolutionary algorithm which is the GA.

This mixing produces a hybrid solution aimed at optimizing the scheduling of SWf. The performance of GA can be enhanced by the incorporation of the solution generated by HEFT into the set of randomly generated solutions making up the initial population. HEFT is chosen because it offers very good scheduling of SWf. This approach tends to speed up the scheduling process to reach the optimum aiming to minimize computational time and cost.

This proposed approach is hybrid because a heuristic model, which is the HEFT, intervenes in the generation of the initial population. This mixture HEFT-GA is in order to obtain a set of optimal solution.

Based on the empirical results obtained from our simulations, we demonstrate that the proposed algorithm performs better than other state-of-the-art strategies to solve WS problem in cloud environments.

The remainder of this paper is organized into the followings sections.

We start by discussing several related works in Sect. 2. While Sect. 3 formulates the scientific workflow scheduling in CC that aims to minimize cost and time while meeting deadline and budget. Section 4 proposes a hybrid approach based on GA and HEFT. Experiments and results are reported in Sect. 5.

2 Related work

Several research studies have been conducted in recent years that analyses the SWf scheduling problem in the field of CC. Therefore, in the literature, several approaches have been applied in order to optimize one or several objectives. Table 1 states a list of criteria, constraints, methods, SWf applications and implementation environment chosen in some research topics. This part is organized in four parts that each one is related to a different optimization problem type.

Table 1 State of the art related to scientific workflow scheduling

Performance metrics	QoS constraints	Environment	Method	Structural	Types of SWfA	References
Makespan	–	Amazon EC2	Compiler optimization	DAG	Montage	[5]
Makespan	–	MATLAB	ACO	DAG	Cybershake SIPHT	[6]
Makespan	–	CLAVIRE	GA	DAG	Complex Wf for flood simulation	[7]
Makespan	Deadline	Cloudism	LEFT GA	DAG	Montage Cybershake Epigenomic SIPHT LIGO	[8]
Makespan	Deadline	–	GA	DAG	Montage Cybershake LIGO	[9]
Makespan	Deadline budget	CloudSim	Deadline constraint scheduling algo	DAG	Montage Cybershake SIPHT LIGO	[10]
Cost	Deadline	IBM ILOG CPLEX	Math Prog	DAG	Montage Cybershake Epigenomic SIPHT LIGO	[11]
Cost	Deadline	Amazon EC2	A* approach	DAG	Montage LIGO Epigenomic	[12]
Makespan cost	–	Amazon EC2	GA	DAG	Montage Cybershake Epigenomic SIPHT LIGO	[13]
Makespan cost	–	–	GA	DAG	Montage Epigenomic Cybershake SIPHT LIGO	[14]
Makespan cost	–	Real platform	Vector ordinal optimization method	DAG	LIGO	[15]
Makespan cost	Deadline	–	GA	DAG	Montage LIGO Cybershake Epigenomic	[16]
Makespan cost	Deadline	Amazon EC2	List scheduling method ACO	DAG	Montage LIGO Cybershake Epigenomic	[17]
Makespan cost	Deadline	CloudSim	PSO	DAG	Montage Cybershake SIPHT LIGO	[18]
Makespan cost	Deadline	MATLAB	Deadline constraint scheduling algo	DAG	Montage Cybershake LIGO SIPHT	[19]
Makespan cost	Deadline budget	–	GA	DAG	Balanced structure unbalanced structure	[20]
Makespan cost	Deadline security	CloudSim	PSO GA	DAG	Cybershake LIGO Epigenomic	[21]
Makespan cost	Stored data	Microsoft Azure	Activity greedy scheduling	DAG	SciEvol	[22]

Table 1 (continued)

Performance metrics	QoS constraints	Environment	Method	Structural	Types of SWfA	References
Makespan resource usage	–	Distributed elastic Wf execution (DEWE)	Wf scheduling optimization	DAG	Montage Cybershake Epigenomic SIPHT LIGO	[23]
Performance gain resource consolidation	–	WorkflowSim	Task clustering technique	DAG	Montage Cybershake LIGO SIPHT	[24]
Memory demands throughput	–	IBM RC2	Ordinal optimization algorithm	DAG	LIGO	[25]
Makespan cost reliability energy	–	ASKALON	List scheduling heuristic	DAG	WIEN2K MeteoAG	[26]
Makespan reliability	–	GridSim	GA	DAG	Random DAG	[27]
Failure tasks Makespan Cost	Deadline	CloudSim	Auto-scaling methods	DAG	Cybershake Epigenomic	[28]
Completed Wf makespan cost	Deadline budget	Cloudsim	Deadline and budget constraints scheduling algo	DAG	Montage Cybershake Epigenomic SIPHT LIGO	[29]
Makespan data dependencies	Load balancing proximity-aware Min CPU speed Min RAM or Hard disk	CycloidGrid	K-way Fiduccia-Mattheyses HEFT	DAG	Montage Cybershake Epigenomic SIPHT	[30]

2.1 Unconstrained SWf scheduling problems

In [5], optimization of scientific workflow execution in clouds by exploiting multicore system with the parallelization of bottleneck tasks is the objective of this paper.

In [23], the authors proposed a solution to optimize the resource usage of a workflow schedule.

Sahni et al. [24] propose a workflow and platform aware task clustering technique that aims to achieve maximum possible parallelism among the tasks.

In [25], Zhang et al. apply an ordinal optimization method iteratively in order to execute scientific workflow on elastic cloud compute nodes with dynamic workload.

In [13], the authors proposed an optimization solution of both makespan and cost.

Xiang et al. [6] presented a novel WS algorithm to minimize makespan in heterogeneous environments.

In [14], the authors optimized the makespan and monetary cost in cloud environment using GA.

In [7], Chirkin et al. proposed an optimization solution of the workflow makespan.

A solution based on list scheduling heuristic approach in order to optimize four objectives such as makespan, cost, energy consumption and reliability is proposed by Fard et al. [26].

In [27], the authors proposed a novel approach based on GA that uses the reliability-driven reputation to optimize makespan and reliability.

Zhang et al. [15] proposed a vectorized ordinal optimization approach to optimize makespan and cost.

2.2 SWf scheduling problems with deadline constraint

In [16], the authors minimized the execution cost of the workflow while meeting the deadline in CC environment.

Vinay et al. [28] scaled resources vertically in order to maximize resources utilization that are required to execute scientific workflow to meet deadline.

The minimization of the execution cost of a workflow in clouds under a deadline constraint was the main objective of [17].

In [8], the authors proposed an heuristic algorithm for scheduling workflows in deadline constrained clouds used for initialization of proposed GA.

In [18], Rodriguez et al. presented a resource provisioning and scheduling strategy for scientific workflow on IaaS.

Li and Cai [11] divide the workflow deadline into task deadlines in order to minimize resource renting costs.

Developing a scheduling system in order to minimize the expected monetary cost given the user specified probabilistic deadline guarantees is the objective addressed in [12].

In [19], the authors proposed a WS strategy in order to minimize the makespan and cost while meeting the deadline.

Visheratin et al. [9] proposed a co-evolutional GA for scheduling series of workflows in order to minimize the makespan while meeting deadline.

2.3 Optimization problems with deadline and budget constraints

Maximization of the number of completed workflow under both budget and deadline constraints is the optimization problem treated in [29].

In [10], Calheiros and Buyya replicated workflow tasks using idle time to mitigate effects of performance variation of resources.

In [20], the authors proposed an algorithm to solve the WS problem in order to minimize the time and cost while meeting deadline and budget.

Shishido et al. [21] used a mixed methodology based on security-aware and cost-aware WS algorithm that applied GA to optimize the combinatorial scheduling scheme.

2.4 Optimization problem with other constraints

The main objective of [30] was the partitioning workflow application into sub-workflow in order to minimize data dependencies and then execute each sub-workflow to minimize their partial makespan.

Liu et al. [22] executed a SWf in a multisite cloud while reducing makespan and cost.

As shown in Fig. 2, the majority of the work targeted on:

- *Performance metrics* makespan and cost.
- *QoS constraints* deadline and to a lesser extent budget.
- *SWf applications* Cybershake, LIGO, Montage, Epigenomic and SIPHT.

After this study, we can deduce that makespan and cost are the most important QoS parameters to study in task scheduling problems in the cloud environment while respecting QoS constraints like deadline and budget. Workflows must be executed before the scheduled deadline while not exceeding the allocated budget. For this, we deduce that the makespan which represents the completion time of the last task of the workflow must be lower than the deadline for a total cost not exceeding the budget. These constraints are defined in Eqs. (6) and (7).

3 Problem formulation

We address a SWf scheduling in CC in order to minimize the computational cost and the execution time (makespan) while meeting two constraints which are the deadline and the budget.

The problem consists to well respond to the client demands to run a workflow which is a set of interdependent tasks.

3.1 Structural representation

The WS representation can only be done using an adequate technique. For this, we choose an oriented graph which does not have a circuit and whose arcs are oriented. This type of graph is known as an acyclic oriented graph [directed acyclic graph (DAG)] [10, 13, 16] which is the most popular model as shown in Table 1. The nodes of the DAG are the tasks, and the arcs represent the dependency relationships between the tasks. Figure 3 shows a general workflow's DAG scheme which contains 8 tasks. Other SWf DAG schemes are shown in Fig. 10.

DAG is represented by a couple of vertex (V) and edge (E). It is usually denoted by $W = \{V, E\}$ where

- $V = \{t_0 \ldots t_n\}$ is a set of n tasks of a SWf;

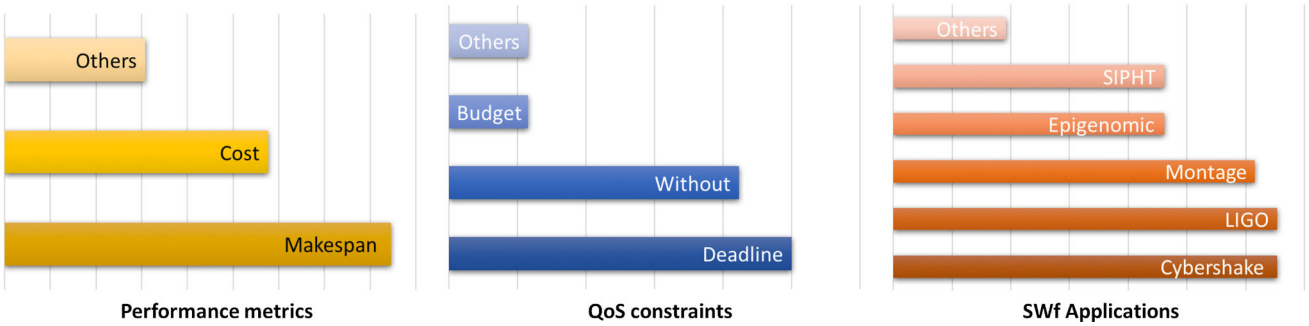

Fig. 2 Comparison between (1) performance metrics, (2) QoS constraints and (3) SWf applications

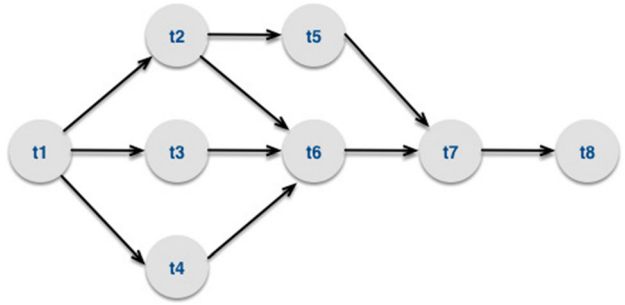

Fig. 3 Workflow DAG scheme sample

- $E = \{t_i \rightarrow t_j | t_i, t_j \in V$ and t_i is parent of $t_j \quad i,j = 1,\ldots,n\}$ is a set of directed edges (control, data dependencies) that connect the tasks (vertices).
- t_1 is the start task denoted t_{start}.
- t_n is the end task denoted t_{end}.

The DAG structure refers to parent–child relationships. We assume that the $Child_i$ task is a successor to the $Parent_i$ task if there is an edge from $Parent_i$ to $Child_i$ in the DAG. Upon task precedence constraint, only if the predecessor $Parent_i$ finishes its execution and sends a message to its successor $Child_i$, the latter can start its execution.

In cloud environment, tasks must be mapped to a set of resources used as a set of virtual machine (VM), in order to be executed. This set denoted $VM = \{vm_0 \ldots vm_m\}$. Each VM has its proper capacity (CPU, $memory$, BW).

DAG is usually represented by a binary square matrix $M[n, n]$.

$$M[i,j] = \begin{cases} 1 & \text{when } t_i \in Parent_j \\ 0 & \text{otherwise} \end{cases}$$

Figure 4 shows the representation of a workflow sample shown schematically in Fig. 3.

3.2 Problem statement

Each task mapped to a particular VM has a computational cost calculated based on a time interval which designates the unit of measurement for calculating costs, especially since we assume in this paper that the resources are billed per unit of time of use. This time interval is called "quantum." Logically, the fastest VM must be more costly. For this, we must find a trade-off between computation time and cost. The notation used in our problem modeling is described in Table 2.

The computational time CT_i^k of t_i executed in vm_k is defined in Eq. (1) as:

$$CT_i^k = \frac{Size_i}{CompC_k} \quad i = 1,\ldots,n \quad k = 1,\ldots,m \tag{1}$$

The communication between two tasks t_i and t_j needs a transfer time from parent t_i to child t_j. Transfer time denoted TrT_{ij} is calculated using the formula of Eq. (2):

$$TrT_{ij} = \frac{Data_{ij}}{BW} \quad i,j = 1,\ldots,n \tag{2}$$

It should be well noted that the TrT_{ij} is $zero$ if t_i and t_j belong to the same VM. Therefore, the internal data transfer is free of cost, which is the case in most of cloud data center.

The $makespan$ expresses the total time spent to complete the user job. $Makespan$ can be defined in Eq. (3) as:

$$Makespan_k = Max\{CT_i^k\} \quad i = 1,\ldots,n \quad k = 1,\ldots,m \tag{3}$$

The objectives functions of this proposed model can be defined in Eqs. (4) and (5) as:

$$MinCost = \sum_{k=1}^{m} \frac{FCT_k - SCT_k}{Quantum} \times UCC_k \tag{4}$$

$$MinMakespan = Max\{Makespan_k\} \quad k = 1,\ldots,m \tag{5}$$

subject to:

$Parent_i$ \ $Child_i$	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
t_1	0	**1**	**1**	**1**	0	0	0	0
t_2	0	0	0	0	**1**	**1**	0	0
t_3	0	0	0	0	0	**1**	0	0
t_4	0	0	0	0	0	**1**	0	0
t_5	0	0	0	0	0	0	**1**	0
t_6	0	0	0	0	0	0	**1**	0
t_7	0	0	0	0	0	0	0	**1**
t_8	0	0	0	0	0	0	0	0

Fig. 4 8×8 Matrix of a workflow example

Table 2 Parameter summary

Parameter	Meaning
CT_i^k	Computational time for t_i in vm_k
$Size_i$	Size of t_i
$CompC_k$	Computational capacity of vm_k ($FLOPS*$)
BW	Average bandwidth
$Data_{ij}$	Data out from $t_i \rightarrow t_j$
SCT_k	Start computation time in vm_k
FCT_k	Finish computation time in vm_k
$Quantum$	Discrete unit to calculate the cost of using VM
UCC_k	Unit computation cost of vm_k
$Makespan_k$	Total time spend to complete tasks in vm_k
$Deadline$	Time till which the tasks should be finished
$Budget$	Total cost not to be exceeded

*FLoating-point OPeration per Second

Task ID	t1	t2	....	tn-1	tn
Vm ID	Vm1	Vm3	....	Vm2	Vm1

Fig. 5 Chromosome encoding

$$Makespan \leq Deadline \tag{6}$$

$$Cost \leq Budget \tag{7}$$

4 Proposed approach

In this paper, we propose a hybrid GA-based approach mixed with HEFT to generate an initial population. In our proposed approach, we are looking for a solution in which the best trade-off *time*/*cost* has been applied while meeting deadline and budget constraints.

4.1 Implementation of approach

1. **Encoding**
   In the literature, different types of encoding representation are proposed [4, 13, 27]. In this paper, we choose to use a direct representation, which is the more adapted to our information encoding. We propose to encode a chromosome as a *n*-sized collection. Each box *i* of the collection contains the *VM* identifier used by t_i. The encoding is represented as shown in Fig. 5. In the proposed example, we identify two major information. The indexes of the vector depict the tasks that are scheduled and the number in each cell identifies the *VM* instance to which the task is allocated.

2. **Initialization**
   Algorithm 1 presents of the different steps taken to generate an initial population to implement our solution based on GA.

3. **Fitness function**
   To solve our problem using GA, we have two goals to achieve. We want to minimize the computational time and the computational cost of executing the workflow. For this, we must clearly define a fitness function that meets this dual objective. The reader must be able to clearly understand how the fitness score is calculated. This function should generate intuitive results. The best solutions must have the best scores while the worst solutions must have the worst scores. First, we need to normalize all the fitness factors that make up our fitness function to be able to define it as defined in Eq. (8). Our function is composed of two factors defined as follows:

$$f1(s) = \frac{Makespan(s)}{Deadline} \quad f2(s) = \frac{Cost(s)}{Budget}$$

   $f1$ as the inverse of the makespan with deadline to ensure that the makespan does not exceed the deadline. $f2$ as the inverse of the cost with budget to ensure that the cost does not exceed the allocated budget. The fitness function is defined as below:

$$\begin{aligned} f(s) &= w \times f1(s) + (1-w) \times f2(s) \\ &\text{subject to} \quad w \in [0,1] \end{aligned} \tag{8}$$

   where

   - $s \in Population$
   - w: the weight of time in the fitness function.
   - $1 - w$: the weight of cost in the fitness function.

4. **Selection operation**
   The tournament selection strategy [16, 19], the most popular selection technique of GA, is used to select the chromosome. In Algorithm 2, a presentation of the different steps was taken to apply this strategy. Figure 6 shows a tournament selection example applied on a sample of chromosomes.

Algorithm 1 Generate an initial population

Input: Empty population and population size $size_p$
Output: Population of $size_p$ chromosomes

1: Generate randomly $size_p - 1$ of candidate solutions
2: Integrate the HEFT solution into the population
3: Calculate the Fitness Value (FV) of each solution of the population
4: Sort the population according to the FVs calculated in Step 3

Algorithm 2 Tournament selection

Input: Population of chromosomes and the tournament size (nbr)
Output: Selected chromosomes

1: Randomly choose nbr chromosomes from the population.
2: **if** crossover **then**
3: The two chromosomes with the highest FV are selected
4: **else if** mutation **then**
5: The chromosome with the highest FV is selected
6: **end if**

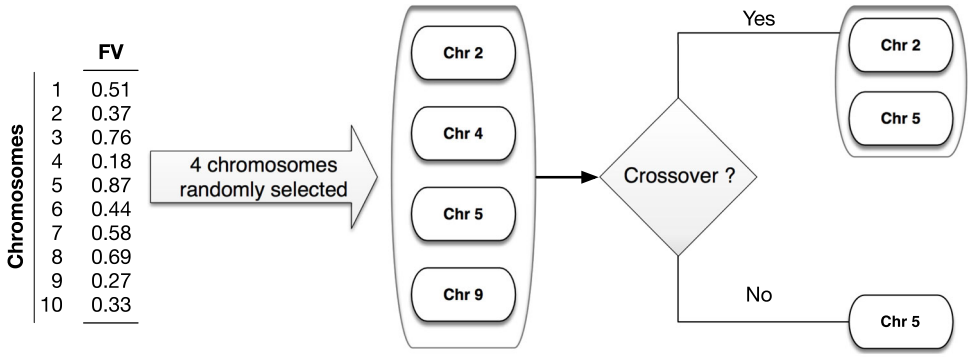

Fig. 6 Tournament selection for the GA

5. **Crossover**
Using the tournament selection method, two chromosomes are selected for a two-point crossover [4, 13] operation wherein alternating segments are swapped to get new offsprings. In other words, these two chromosomes chosen will give rise to two offsprings after a crossing. For example, Fig. 7 shows a two-point crossover example.

6. **Mutation**
We propose to use an integer representation. A random task from the set of workflow tasks is assigned to a randomly chosen *VM* [4] as shown in Fig. 8.

5 Simulation environment

5.1 Simulation setup

We have generated synthetic workflow data obtained from the Pegasus workflow repository [31]. To study the effectiveness of proposed SWf scheduling algorithm, we have applied simulation parameters summarized in Table 3.

5.2 Framework simulation environment

To evaluate our proposed solution, we chose to use WorkflowSim Framework based on CloudSim [18, 29]. Figure 9 explains the different steps from the introduction of the DAX[1] file generated by Pegasus workflow repository to the matching of workflow *tasks* to *VM* before being runned.

A *workflow planner* generates a list of tasks that are introduced first in raw state as an XML[2] file. The *workflow parser* module intervenes to prepare this tasks list. Tasks can be grouped in a set of jobs by the *clustering engine* if necessary. After that, these tasks must be ordered by the *workflow engine* taking into account the dependency constraints. At this level, the *workflow scheduler* intervenes to match ordered *tasks* to available *VMs* before being runned by *processors*.

6 Implementation and results

Following an in-depth study of the related work, we identify several areas of applications related to the scheduling problem. We examine five families of SWf applications as Montage, Cybershake, Epigenomics, LIGO and SIPHT, which are abstractions of dataflows that are used in real applications. These SWf applications were chosen because they represent a wide range of application domains and a variety of resource requirements. On this

[1] Description of an abstract workflow in eXtended Markup Language (XML) format that is used as the primary input into Pegasus.
[2] General text-oriented document format.

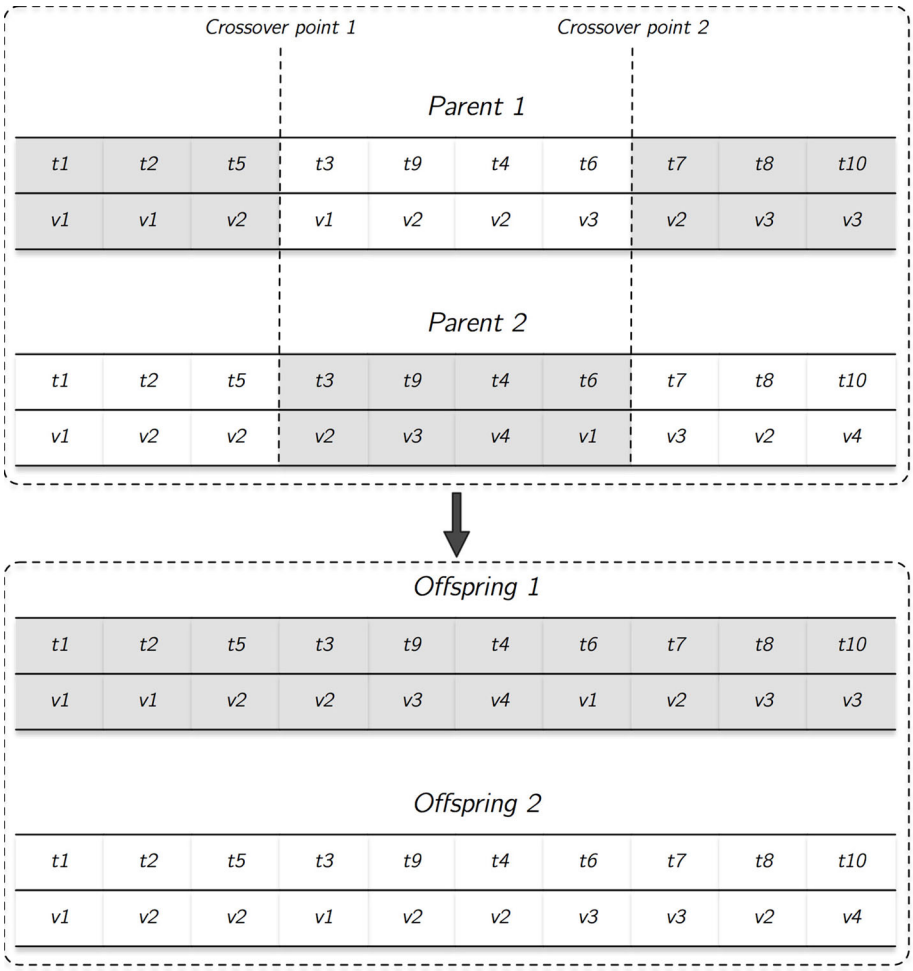

Fig. 7 An example of the two-point crossover

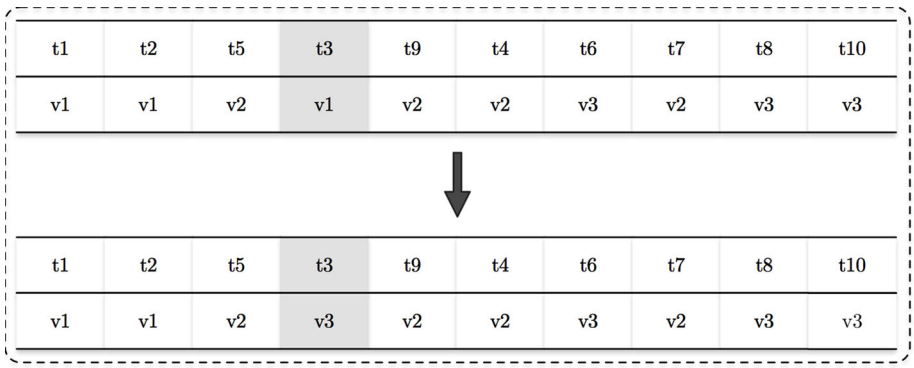

Fig. 8 The mutation operator

Table 3 Workflow scheduling simulation parameters

Datacenter parameters		VMs parameters		Cost parameters	
Number of VMs	20	RAM (MB)	512	Processing usage cost	3.0
Number of cloud users	1	MIPS	1000	Memory usage cost	0.05
		BW	1000	Storage usage cost	0.1
		Pes Number	1	BW usage cost	0.1

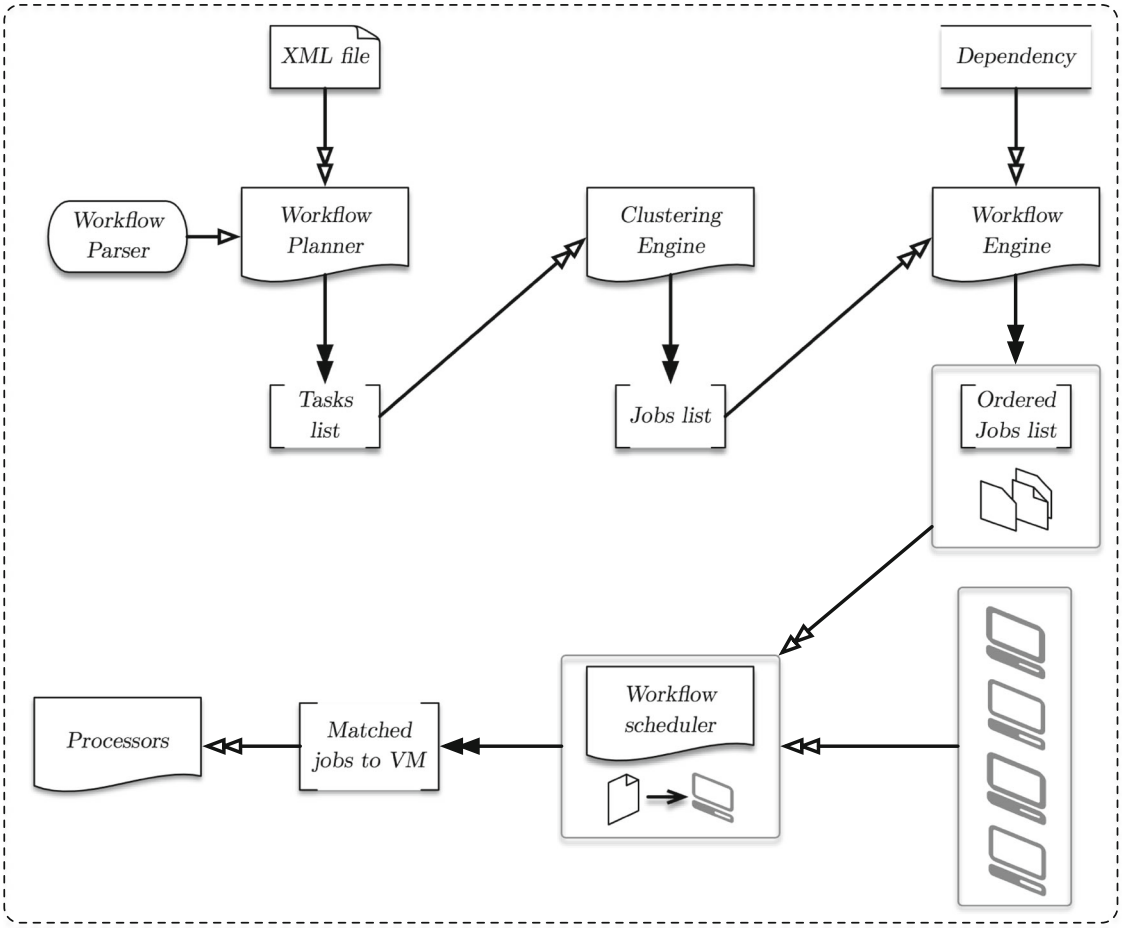

Fig. 9 WorkflowSim framework components

basis, Table 4 shows a comparison between these SWf applications in terms of system intensiveness while their DAGs are represented in Fig. 10.

We conducted a series of experiments using existing heuristic algorithms such as:

- FCFS [19]
- MinMin [7]
- MaxMin [20, 27]
- RoundRobin [30]

Table 4 Comparison between the existing SWf applications

SWf application	Domain	System intensiveness
Montage	Astronomy	Data-intensive
CyberShake	Earthquake science	Data/memory-intensive
Epigenomics	Bioinformatics	CPU-intensive
LIGO	Physics	CPU-intensive
SIPHT	Bioinformatics	CPU-intensive

- HEFT

We managed all jobs with the Pegasus WfMS, which transforms high-level descriptions of workflows into specific sequences of operations and identifies the computing resources required for execution. Here are the results:

- The results of a series of experimentations applied to Montage workflow to calculate the makespan are described in Table 5.
- The result of a series of experimentations applied to Cybershake workflow, with, respectively, 100 and 1000 nodes, to calculate the makespan is described in Table 6.
- The result of a series of experimentations applied to Epigenomics workflow to calculate the makespan is described in Table 7.
- The result of a series of experimentations applied to LIGO workflow to calculate the makespan is shown in Table 8.

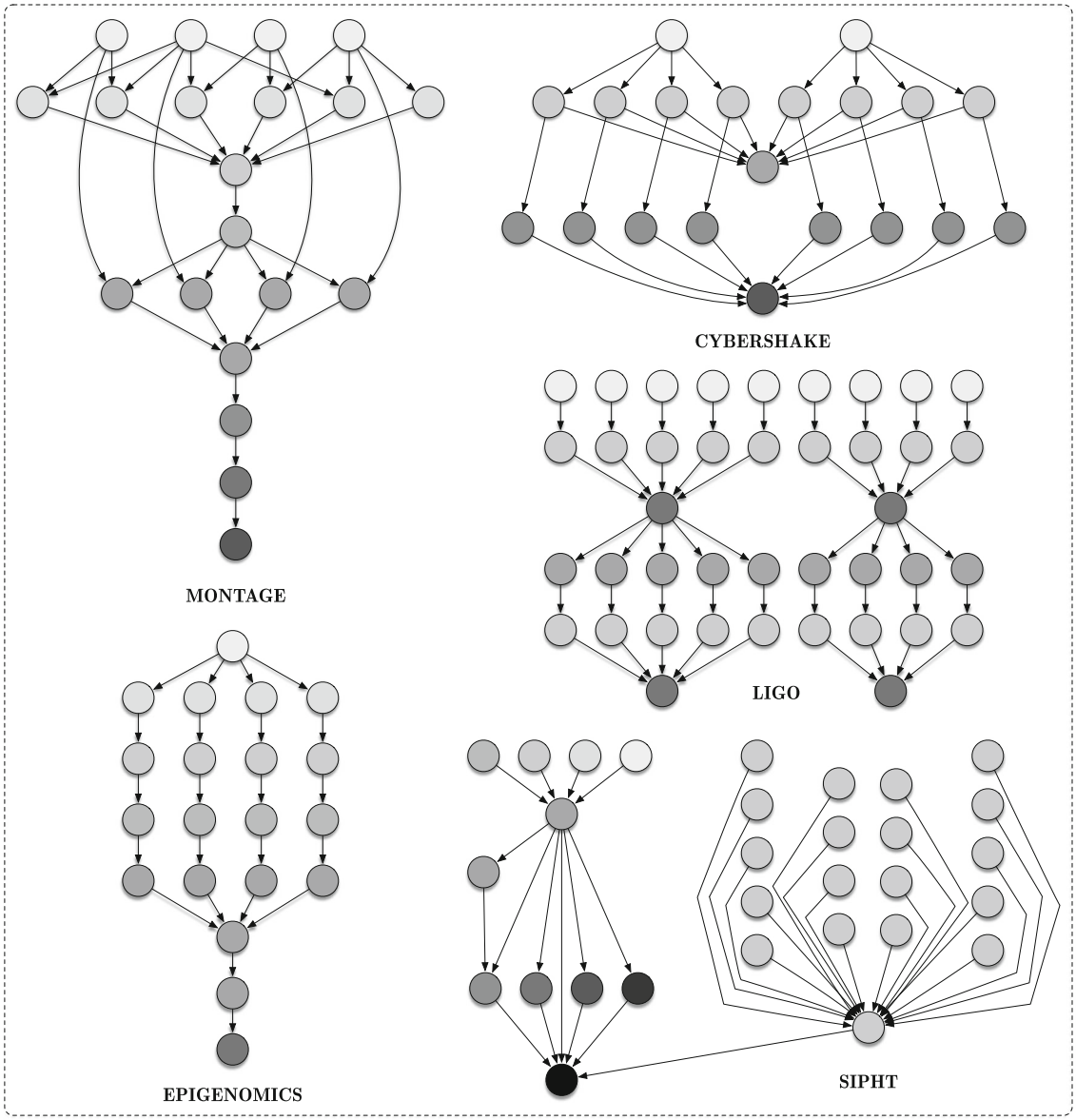

Fig. 10 Real-world scientific workflows DAGs

Table 5 Experimental results for montage datasets of 100 and 1000 tasks

Scheduling algorithm	Makespan		Cost	
	Montage_100	Montage_1000	Montage_100	Montage_1000
FCFS	103.45	911.96	3451.96	36,260.73
MinMin	103.53	912.38	3453.14	36,258.04
MaxMin	103.19	912.2	3452.34	36,264.65
RoundRobin	103.44	912.56	3451.71	36,263.45
HEFT	103.4	912.36	3449.3	36,265.84
HEFTGA	**103.12**	**912.02**	**3449.21**	**36,260.11**

Best experimentation values for each scientific workflow are given in bold

- The result of a series of experimentations applied to SIPHT workflow to calculate the makespan is shown in Table 9.

Figures 11 and 12 show the experimental results of the makespan, while Figs. 13 and 14 show the experimental results of the cost.

Table 6 Experimental results for cybershake datasets of 100 and 1000 tasks

Scheduling algorithm	Makespan		Cost	
	Cybershake_100	Cybershake_1000	Cybershake_100	Cybershake_1000
FCFS	321.92	1284.87	20,126.31	100,196.92
MinMin	325.56	1309.81	20,114.81	100,193.49
MaxMin	**305.19**	**1272.49**	20,126.6	100,208.38
RoundRobin	321.92	1310.44	20,126.16	**100,192.11**
HEFT	415.92	1430.64	20,075.22	100,194.66
HEFTGA	323.19	1310.31	**20,057.12**	100,193.27

Best experimentation values for each scientific workflow are given in bold

Table 7 Experimental results for Epigenomics datasets of 100 and 997 tasks

Scheduling algorithm	Makespan		Cost	
	Epigenomics_100	Epigenomics_997	Epigenomics_100	Epigenomics_997
FCFS	34,988.53	208,608.52	1,217,246.7	11,627,118.97
MinMin	43,044.26	213,651.11	1,217,246.32	11,620,692.01
MaxMin	36,972.03	210,441.58	1,217,246.04	11,651,559.76
RoundRobin	40,036.51	208,632.38	1,217,246.83	11,620,725.44
HEFT	32,849.6	217,396.71	1,217,241.77	11,758,522.58
HEFTGA	**31,766.09**	**206,321.36**	**1,217,241.33**	**11,611,331.77**

Best experimentation values for each scientific workflow are given in bold

Table 8 Experimental results for LIGO datasets of 100 and 1000 tasks

Scheduling algorithm	Makespan		Cost	
	LIGO_100	LIGO_1000	LIGO_100	LIGO_1000
FCFS	**1519.72**	11,759.27	63,468.8	687,170.27
MinMin	1762.86	11,791.02	63,475.28	687,160.99
MaxMin	1523.58	11,663.18	63,476	687,145.08
RoundRobin	1519.9	11,815.65	63,476.72	687,151.86
HEFT	1901.72	12,902.44	63,460.63	687,149.56
HEFTGA	1616.01	**11,223.01**	**63,442.01**	**687,107.47**

Best experimentation values for each scientific workflow are given in bold

Table 9 Experimental results for the SIPHT datasets of 100 and 1000 tasks

Scheduling algorithm	Makespan		Cost	
	SIPHT_100	SIPHT_1000	SIPHT_100	SIPHT_1000
FCFS	4478.22	9645.15	52,216.12	521,644.86
MinMin	4479.07	11,630.67	**52,214.47**	521,658.18
MaxMin	4475.62	**9621.08**	52,217.05	521,647.71
RoundRobin	4478.22	11,121.93	52,215.94	521,649.38
HEFT	4475.62	10,827.18	52,215.81	521,644.29
HEFTGA	**4475.12**	10,827.18	52,215.81	**521,622.09**

Best experimentation values for each scientific workflow are given in bold

The proposed approach HEFTGA that we applied is based on the population of 20 chromosomes transformed in 100 generations which is a sufficient number to achieve a good convergence rate.

This proposed algorithm is developed in Netbeans 8.1 using Java programming language.

All experiments were performed on a computer with 2.4 GHz Intel Core i5 CPU and 8Go 1333 MHz of RAM.

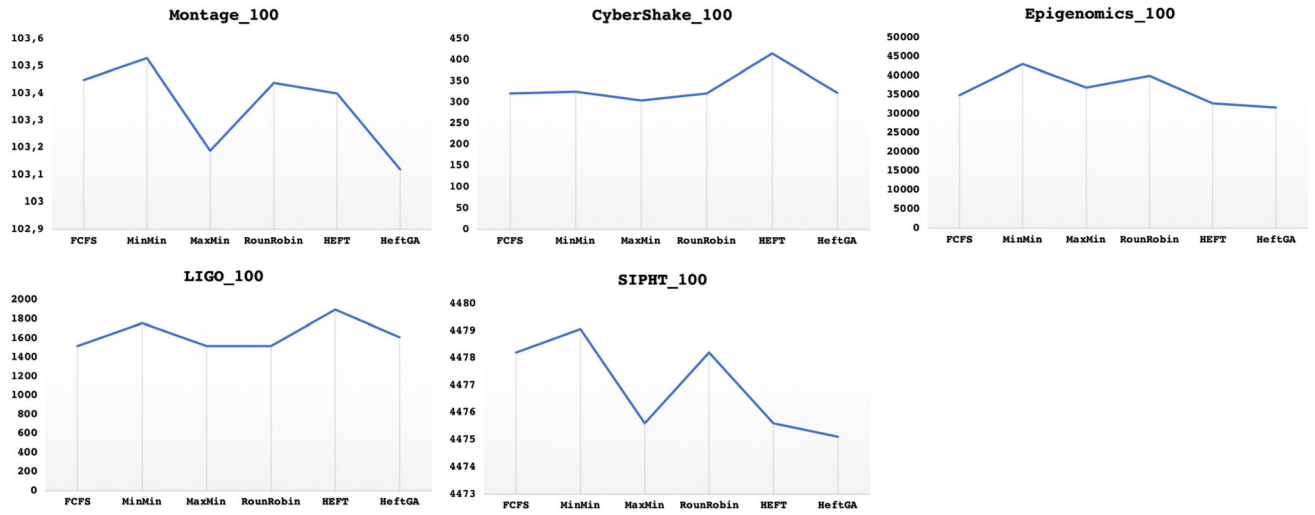

Fig. 11 Simulation results plot of the makespan for 100 tasks and 20 VMs

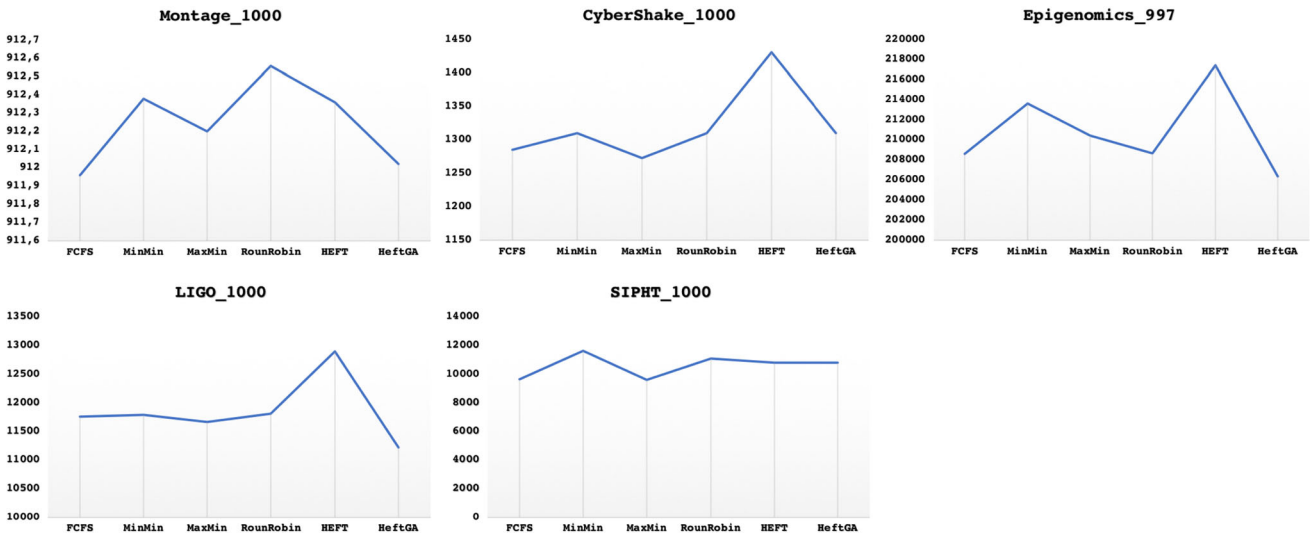

Fig. 12 Simulation results plot of the makespan for 1000 tasks and 20 VMs

Experimental results presented in Figs. 10, 11, 12 and 13 show that HEFTGA outperforms other state-of-the-art WS strategies in most cases.

- Regarding Montage, we note that our solution gives better results in different cases. For a workflow consisting of 100 tasks (respectively, 1000), HEFTGA completes their execution after 103.12 s (respectively, 912.02) for a cost of 3449.21 (respectively, 36,260.11), whereas HEFT executes them with a cost higher in more time.
- For Cybershake, the simulation results are less interesting when compared with those realized with Montage, but still they are better than those of HEFT. Workflow composed of 100 tasks is performed at a lower cost than others heuristics.
- Unlike Cybershake, Epigenomics gives very good results to complete the execution of different tasks like Montage. The trade-off between time and cost is realized successfully.
- About LIGO, the two workflows composed of 100 and 1000 tasks are completed with the cheapest costs by comparing them with the other results obtained by applying the other heuristics. A very good compromise makespan cost is realized for the second workflow consisting of 1000 tasks while the time is moderately acceptable for 100 tasks.
- The results obtained with SIPHT are very interesting for the makespan of the workflow composed of 100 tasks and the completion cost of 1000 tasks while the makespan of 1000 tasks and the cost of 100 tasks are just acceptable.

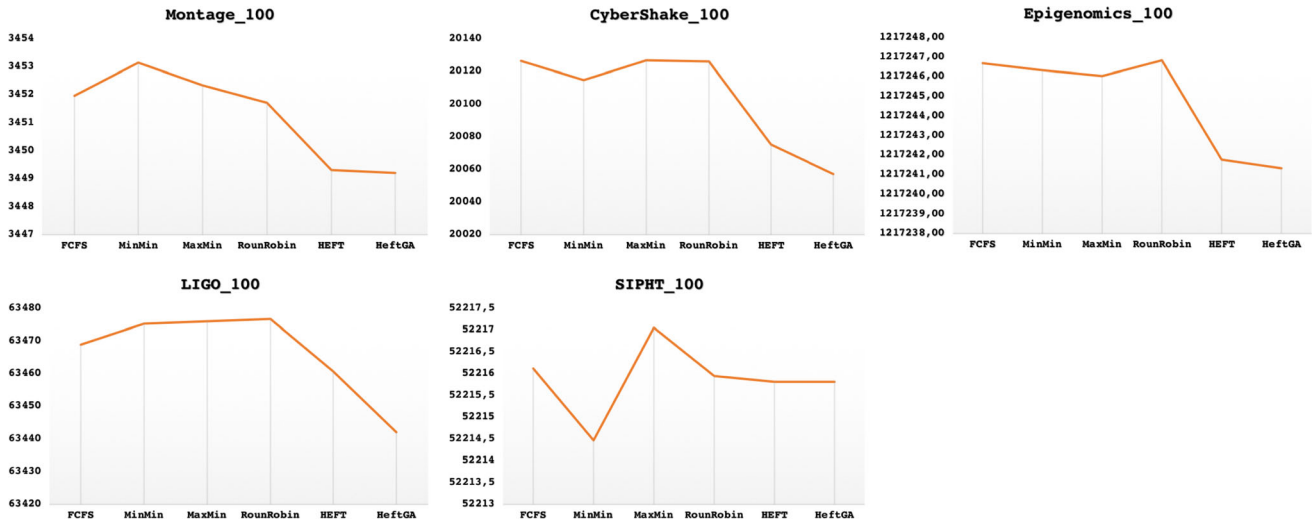

Fig. 13 Simulation results plot of the cost for 100 tasks and 20 VMs

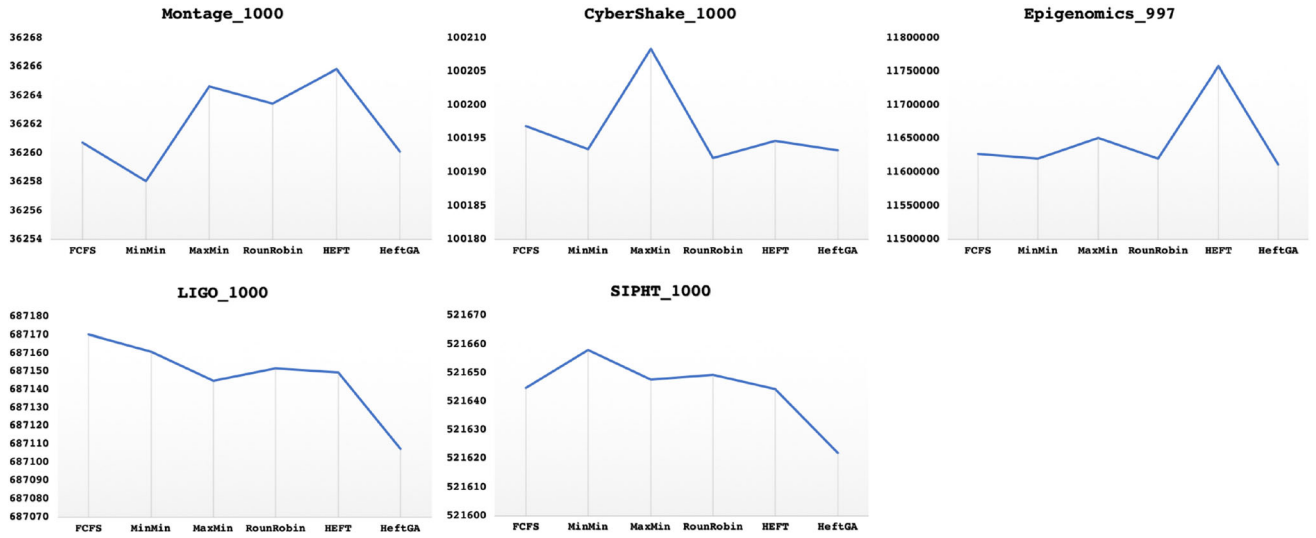

Fig. 14 Simulation results plot of the cost for 1000 tasks and 20 VMs

To conclude, we can deduce that HEFTGA is well suited to Montage and Epigenomics applications since there is a successful compromise between time and cost. On the other hand, we can see that for Cybershake workflow profit is not very big. In addition, it should be mentioned that HEFTGA provides best results that HEFT in all cases for all workflow applications. This is due to the integration of the solution generated by HEFT into the initial population of our HEFTGA approach, which gives rise to a hybrid solution that guarantees the results of HEFT in the worst case otherwise much more efficient results.

7 Conclusion

In this paper, a cloud hybrid evolutionary approach for deadline and budget constrained real-world scientific workflow scheduling is proposed. As a solution of this problem, the proposed algorithm integrates the HEFT solution into the initial population used by our approach to achieve an optimal execution time and minimal execution cost. We have presented the different objectives as multiple QoS function including time and cost. Experiments on real-world SWf as Montage, Cybershake, Epigenomics, LIGO and SIPHT show that our approach HEFTGA has outperformed several state-of-the-art algorithm as FCFS,

MinMin, MaxMin and RoundRobin that were previously used to solve the WS problem.

In future work, we plan to deal with the problem of data centers power consumption when planning workflows in cloud environments. Another work ahead is to simulate SWf in heterogeneous cloud environments globally distributed, hence the importance of taking into account the time and cost of data transfer between different data centers.

Compliance with ethical standards

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

References

1. Poola D, Ramamohanarao K, Buyya R (2014) Fault-tolerant workflow scheduling using spot instances on clouds. Procedia Comput Sci 29(12):523–533. https://doi.org/10.1016/j.procs.2014.05.047
2. Wang J, Abdelbaky M, Diaz-Montes J, Purawat S, Parashar M, Altintas I (2016) Kepler + cometcloud: dynamic scientific workflow execution on federated cloud resources. Procedia Comput Sci 80(12):700–711. https://doi.org/10.1016/j.procs.2016.05.363
3. Alkhanak EN, Lee SP, Rezaei R, Parizi RM (2016) Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: a review, classifications, and open issues. J Syst Softw 113:1–26. https://doi.org/10.1016/j.jss.2015.11.023
4. Aziza H, Krichen S (2018) Bi-objective decision support system for task-scheduling based on genetic algorithm in cloud computing. Computing 100(2):65–91. https://doi.org/10.1007/s00607-017-0566-5
5. Jiang Q, Lee YC, Arenaz M, Leslie LM, Zomaya AY (2014) Optimizing scientific workflows in the cloud: a montage example. In: 2014 IEEE/ACM 7th international conference on utility and cloud computing, December 2014, pp 517–522. https://doi.org/10.1109/UCC.2014.77
6. Xiang B, Zhang B, Zhang L (2017) Greedy-ant: ant colony system-inspired workflow scheduling for heterogeneous computing. IEEE Access 5:11404–11412. https://doi.org/10.1109/ACCESS.2017.2715279
7. Chirkin AM, Belloum ASZ, Kovalchuk SV, Makkes MX, Melnik MA, Visheratin AA, Nasonov DA (2017) Execution time estimation for workflow scheduling. Future Gener Comput Syst 75:376–387. https://doi.org/10.1016/j.future.2017.01.011
8. Visheratin AA, Melnik M, Nasonov D (2016) Workflow scheduling algorithms for hard-deadline constrained cloud environments. Procedia Comput Sci 80:2098–2106. https://doi.org/10.1016/j.procs.2016.05.529 International conference on computational science 2016, ICCS 2016, 6–8 June 2016, San Diego, California, USA
9. Visheratin A, Melnik M, Butakov N, Nasonov D (2015) Hard-deadline constrained workflows scheduling using metaheuristic algorithms. Procedia Comput Sci 66:506–514. https://doi.org/10.1016/j.procs.2015.11.057 4th international young scientist conference on computational science
10. Calheiros RN, Buyya R (2014) Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans Parallel Distrib Syst 25(7):1787–1796. https://doi.org/10.1109/TPDS.2013.238
11. Li X, Cai Z (2015) Elastic resource provisioning for cloud workflow applications. IEEE Trans Autom Sci Eng 14(1–16):12. https://doi.org/10.1109/TASE.2015.2500574
12. Zhou AC, He B, Liu C (2016) Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds. IEEE Trans Cloud Comput 4(1):34–48. https://doi.org/10.1109/TCC.2015.2404807
13. Zhu Z, Zhang G, Li M, Liu X (2016) Evolutionary multi-objective workflow scheduling in cloud. IEEE Trans Parallel Distrib Syst 27(5):1344–1357. https://doi.org/10.1109/TPDS.2015.2446459
14. Casas I, Taheri J, Ranjan R, Wang L, Zomaya AY (2018) GA-ETI: an enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. J Comput Sci 26:318–331. https://doi.org/10.1016/j.jocs.2016.08.007
15. Zhang F, Cao J, Li K, Khan SU, Hwang K (2014) Multi-objective scheduling of many tasks in cloud platforms. Future Gener Comput Syst 37:309–320. https://doi.org/10.1016/j.future.2013.09.006 Special section: innovative methods and algorithms for advanced data-intensive computing. Special section: semantics, intelligent processing and services for big data. Special section: advances in data-intensive modelling and simulation. Special section: hybrid intelligence for growing internet and its applications
16. Meena J, Kumar M, Vardhan M (2016) Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint. IEEE Access 4:5065–5082. https://doi.org/10.1109/ACCESS.2016.2593903
17. Wu Q, Ishikawa F, Zhu Q, Xia Y, Wen J (2017) Deadline-constrained cost optimization approaches for workflow scheduling in clouds. IEEE Trans Parallel Distrib Syst 28(12):3401–3412. https://doi.org/10.1109/TPDS.2017.2735400
18. Rodriguez MA, Buyya R (2014) Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans Cloud Comput 2(2):222–235. https://doi.org/10.1109/TCC.2014.2314655
19. Haidri RA, Katti CP, Saxen PC (2017) Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing. J King Saud Univ Comput Inf Sci. https://doi.org/10.1016/j.jksuci.2017.10.009
20. Gharooni-fard G, Moein-darbari F, Deldari H, Morvaridi A (2010) Scheduling of scientific workflows using a chaos-genetic algorithm. Procedia Comput Sci 1(1):1445–1454. https://doi.org/10.1016/j.procs.2010.04.160. ICCS 2010
21. Shishido HY, Estrella JC, Toledo CFM, Arantes MS (2018) Genetic-based algorithms applied to a workflow scheduling algorithm with security and deadline constraints in clouds. Comput Electr Eng 69:378–394. https://doi.org/10.1016/j.compeleceng.2017.12.004
22. Ghafarian T, Javadi B, Buyya R (2016) Multi-objective scheduling of scientific workflows in multisite clouds. Future Gener Comput Syst 63:76–95. https://doi.org/10.1016/j.future.2016.04.014 Modeling and management for big data analytics and visualization
23. Lee YC, Han H, Zomaya AY, Yousif M (2015) Resource-efficient workflow scheduling in clouds. Knowl Based Syst 80:153–162. https://doi.org/10.1016/j.knosys.2015.02.012 25th anniversary of knowledge-based systems
24. Sahni J, Vidyarthi DP (2016) Workflow-and-platform aware task clustering for scientific workflow execution in cloud environment. Future Gener Comput Syst 64:61–74. https://doi.org/10.1016/j.future.2016.05.008

25. Zhang F, Cao J, Hwang K, Li K, Khan SU (2015) Adaptive workflow scheduling on cloud computing platforms with iterative ordinal optimization. IEEE Trans Cloud Comput 3(2):156–168. https://doi.org/10.1109/TCC.2014.2350490
26. Fard HM, Prodan R, Fahringer T (2014) Multi-objective list scheduling of workflow applications in distributed computing infrastructures. J Parallel Distrib Comput 74(3):2152–2165. https://doi.org/10.1016/j.jpdc.2013.12.004
27. Wang X, Yeo CS, Buyya R, Jinshu S (2011) Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. Future Gener Comput Syst 27(8):1124–1134. https://doi.org/10.1016/j.future.2011.03.008
28. Vinay K, Dilip Kumar SM (2016) Auto-scaling for deadline constrained scientific workflows in cloud environment. In: 2016 IEEE annual India conference (INDICON), December 2016, pp 1–6. https://doi.org/10.1109/INDICON.2016.7838908
29. Malawski M, Juve G, Deelman E, Nabrzyski J (2015) Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. Future Gener Comput Syst 48:1–18. https://doi.org/10.1016/j.future.2015.01.004 Special section: business and industry specific cloud
30. Ghafarian T, Javadi B, Buyya R (2015) Decentralised workflow scheduling in volunteer computing systems. Int J Parallel Emerg Distrib Syst 30(5):343–365. https://doi.org/10.1080/17445760.2014.973876
31. Workflow management system (2018) https://pegasus.isi.edu/