Budget and SLA Aware Dynamic Workflow Scheduling in Cloud Computing with Heterogeneous Resources

Yifan Yang, Gang Chen, Hui Ma, Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington
Wellington 6140, New Zealand
{yifan.yang, aaron.chen, hui.ma, mengjie.zhang}@ecs.vuw.ac.nz

Victoria Huang
School of Computing & Mathematical Sciences
University of Waikato
Hamilton 3240, New Zealand
guiying.huang@waikato.ac.nz

2021 IEEE Congress on Evolutionary Computation (CEC) | 978-1-7281-8393-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/CEC45853.2021.9504709

***Abstract*—Workflow with different patterns and sizes arrive at a cloud data center dynamically to be processed at virtual machines in the data center, with the aim to minimize overall cost and makespan while satisfying Service Level Agreement (SLA) requirement. To efficiently schedule workflows, manually designed heuristics are proposed in the literature. However, it is time consuming to manually design heuristics. The designed heuristics may not work effectively for heterogeneous workflow since only simple problem related factors are considered in the heuristics. Further, most of the existing approaches ignore the deadline constraints set in SLAs. Genetic Programming Hyper Heuristic (GPHH) can be used to automatically design heuristics for scheduling problems. In this paper, we propose a GPHH approach to automatically generate heuristics for the dynamic workflow scheduling problem, with the goal of minimizing the VM rental fees and SLA penalties. Experiments have been conducted to evaluate the performance of the proposed approach. Compared with several existing heuristics and conventional Genetic Programming (GP) approaches, the proposed Dynamic Workflow Scheduling Genetic Programming (DWSGP) has better performance and is highly adaptable to variations in cloud environment.**

***Index Terms*—dynamic workflow scheduling, cloud computing, genetic programming**

I. Introduction

Scientific workflows are high-level abstractions of scientific applications, consisting of a set of tasks that are linked according to their dependencies [1]. They help to formalize complex scientific processes. Processing these workflows in a reasonable amount of time requires substantial computational resources, such as parallel processing clusters and huge storage space [2]. Therefore, cloud computing technologies are widely utilized for workflow execution. In particular, all tasks in a scientific workflow can be scheduled to execute on one or more virtual machine (VM) instances in cloud. However, depending on how tasks are allocated to VM instances, the cost induced due to violating workflow deadline and the cost spent for renting VM instances can vary significantly. Therefore, workflow scheduling is deemed a challenging problem in cloud computing.

Workflow scheduling is a process that maps and manages the interdependent tasks on the distributed cloud resources [3], [4], in order to satisfy some important objectives, such as minimizing the makespan, budget, deadline, and load balancing. Many existing works studied the workflow scheduling problem on the Infrastructure as a Service (IaaS) based cloud [5], [6], [7], while studies from the perspective of Software as a Service (SaaS) are much more limited. However, as workflows continue to increase in complexity and scale, more and more workflows are outsourced to SaaS providers that help users manage workflow execution. Wu *et al.* [8] considers a SaaS cloud provider that offers workflow execution services to users, as illustrated in Fig. 1. The SaaS provider decides on behalf of users the number and type of VM instances, and assign each task in the workflows to the selected VMs for execution.

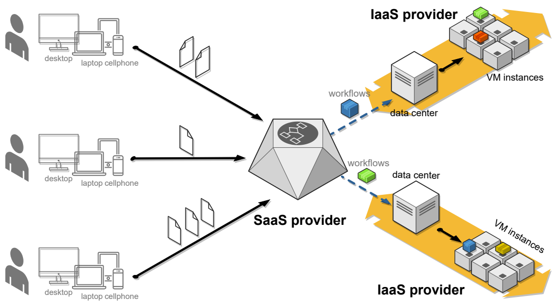

Fig. 1. Workflow scheduling and execution in a SaaS cloud.

SLAs are often established between users and the SaaS provider to specify workflow deadline constraints. The SaaS provider promises to complete the submitted workflows before a deadline, otherwise it will pay the user a certain penalty fee to compensate for their loss. We often do not know a prior the patterns, sizes, and arrival time of workflows. To effectively schedule dynamically arriving workflows, we need to define and study the budget and SLA aware Dynamic Workflow

Scheduling (BSDWS) problem with heterogeneous resources in this paper.

Similar to [5], [4], the BSDWS problem is an NP-hard problem. Many previously developed heuristics and meta-heuristics techniques, such as Heterogeneous Earliest Finish Time (HEFT) [9], Particle Swarm Optimization (PSO) [10], and Ant Colony Algorithm (ACO) [8] cannot be directly applied to solve this problem. Firstly, a majority of existing heuristics were designed for static workflow scheduling [11], [12]. Secondly, many approaches cannot effectively handle different types of workflows and VMs [5], [6]. Thirdly, some dynamic scheduling studies focus only on optimizing a single objective, such as the total execution time or budget[4]. As far as we know, there is no algorithm specifically design to solve the BSDWS problem.

Genetic Programming (GP) is an evolutionary computation (EC) technique that automatically generates solutions without requiring the user to conceive the form or structure of the solution in advance [13]. It is an effective heuristic search technique that has been widely used on many combinatorial optimization problems, such as Job Shop Scheduling (JSS) [14], Arc Routing Problem (ARP) [15]. Although GPHH has been studied on many relevant problems in the past, there are several technical challenges of using GP to solve the BSDWS problem. The first challenge is to identify a suitable collection of terminal nodes and function nodes. The second challenge is to design an accurate process to evaluate the fitness of every GP tree. The third challenge is to determine when to rent a new VM and when to use existing VMs using a GP tree. In view of these challenges, existing GP algorithms proposed in [16], [17] cannot be applied directly to the BSDWS problem.

To solve the BSDWS problem, we propose an new GP algorithm called DWSGP to evolve optimal scheduling heuristic with the objective of minimizing the SLA penalties and VM rental fees. The generated scheduling heuristic allocates each workflow task to one VM instance with the highest priority. By using a newly developed cloud simulator and properly selecting the training problem instances, our DWSGP approach is able to handle different types of workflows and VM instances. The main contributions of this paper are as follows:

1) We formally define the BSDWS problem with the goal to jointly minimize the VM rental fees and the SLA violation penalties from the view of a SaaS provider. Our problem formulation is further supported by a cloud simulator for fitness evaluation.
2) We develop a new algorithm, DWSGP, to evolve competitive and effective scheduling rules, and introduce several new terminal node types for building GP trees.
3) We conduct experimental evaluation using a range of heterogeneous workflows, with different workflow patterns, sizes and arrival time. The results are compared with several existing algorithms under different deadline relaxation factors.

II. Related Work

Workflow scheduling problem is gaining increasing attention in the literature [4], [12], especially in the cloud computing domain. This section summarizes the previous research from the perspective of problem types and solution methods.

A. Workflow Scheduling

The workflow scheduling problem can be divided into static problems and dynamic problems. The former focuses on processing a single workflow in order to achieve certain goals, such as minimizing the VM rental costs, or reducing the makespan. Malawski *et al.* [7] further considered the budget and deadline constraints. Zhou and He [12] designed a transformation-based workflow optimization system.

Different from static workflow scheduling, dynamic scheduling can adaptively process a series of dynamically arriving workflows. Specifically, dynamic workflow scheduling assigns a task to a VM instance based on real-time information [4], [18]. Whenever there is a task pending for allocation, a dynamic scheduling rule will select a VM instance to process it. Liu *et al.* [4] proposed an NOSF approach to minimize the VM rental fees under a deadline constraint where the workflow arrival time is unknown and the task execution time is random. Zeng *et al.* [18] developed a Security-Aware and Budget-Aware workflow scheduling strategy (SABA) to minimize the workflow execution time under the restriction of budget and security requirement.

Only a few research works have explored dynamic workflow scheduling problems so far. Most of these studies target at minimizing the VM rental fees. Compared with the existing studies, we consider for the first time a practical scenario with unknown workflow pattern, workflow size, and workflow arrival time driven by the goal to properly balance deadline violation cost and VM rental cost.

B. Optimization Methods

Since workflow scheduling problems are NP-hard [5], [4], as workflows continue to increase in size and complexity, it is more suitable to solve these problems by using heuristic algorithms.

Several heuristics have been proposed in the literature [2], [5], [8]. HEFT is a classic heuristic to solve workflow scheduling problems [9] with the goal of minimizing the earliest finish time. Modified HEFT can also be found in [5]. In [19], a greedy algorithm is applied to choose the cheapest available resource with the goal of minimizing the total cost. Apart from that, meta-heuristics are also widely applied, such as Particle Swarm Optimization (PSO) [2], [10], Genetic Algorithm (GA) [20], [21] and Ant Colony Algorithm (ACO) [8], [22]. Note that these heuristics are designed manually by human expertise. The design process is time consuming and the designed heuristics may not work effectively across many different problem instances.

Hyper-heuristic is an automated method that generates heuristic rules to solve complicated problems. We use heuristic or rule interchangeably in the remaining of the paper. One of the most popular hyper-heuristic approaches is GP [16], [23], which has been widely demonstrated to be effective at solving various combinatorial optimization problems, such as JSS [14], ARP [15], etc. Yu *et al.* [16] used GP to address the static scheduling problem to minimize the makespan and total cost. In [17], a dynamic problem is presented with uncertain workflow arrival time, but the workflow pattern in each experiment remains constants. Several other previous works scheduled tasks from the perspective of IaaS users. In this paper, we study a different problem of scheduling dynamic workflow execution in SaaS [4], [17].

III. PROBLEM DESCRIPTION

This section formulates a dynamic workflow scheduling problem in cloud computing with the objective to minimize the total cost of deadline violations and VM rental fees.

TABLE I
NOMENCLATURE

Notation	Description
W_i	The i-th workflow
T_{ij}	The j-th task in the i-th workflow
V_k	The k-th virtual machine instance
$Precede(T_{ij})$	The set of adjacent predecessor tasks of T_{ij}
$Succeed(T_{ij})$	The set of adjacent successor tasks of T_{ij}
TS_{ij}	The size of task T_{ij} in W_i
AT_i	The arrival time of workflow W_i
DL_i	The deadline of workflow W_i
$TYPE_k$	The VM type of V_k
CAP_k	The computing capacity of V_k
$PRICE_k$	The rental fee of V_k per hour
RT_{ij}	The ready time of task T_{ij}
FT_{ij}^k	The finish time of task T_{ij} on V_k
$Pre(T_{ij})$	The adjacent task processed before T_{ij} on a VM instance
EP_{ij}	The actual execution time of T_{ij}
REP_{ij}	The relative finish time period of T_{ij}
MS_i	The makespan of W_i
$minMS_i$	The shortest makespan of W_i without waiting time
ξ	The relaxation factor of deadline calculation
δ	The penalty coefficient of the deadline penalty cost
DPC_i	The deadline penalty cost of W_i
$RVMS$	The set of the rent VMs
T_{first}^k	The first task processed on V_k
T_{last}^k	The last task processed on V_k
LP_k	The rent period of V_k
RF_k	The rental fee of the k-th VM.

A. Problem Overview

A workflow W_i is generally represented as a Directed Acyclic Graph $DAG = (V, A)$. V is a set of vertices denoting different $\{T_{ij}\}$ as shown in Fig. 2. A is a set of directed edges, where each edge connects a predecessor task to one of its successor task to enforce their execution order. Specifically, any *successor task*, denoted as *Succeed(T_{ij})*, can only be executed when all of its *Predecessor tasks*, denotes as *Precede(T_{ij})*, are completed. A task can have multiple predecessor (or successor) tasks and can be the predecessor (or successor) of multiple tasks. For example, in Fig. 2, $Precede(\mathrm{T}_{i2})$ = $\{\mathrm{T}_{i1}\}$ and $Succeed(T_{i2}) = \{T_{i3}, T_{i4}, T_{i5}\}$. In particular, a task without predecessors is an *entry task*, and a task without successors is an $exit\ task$, such as T_{i1} and T_{i9}. Each workflow has its own inherent features to distinguish it from other workflows, which can be captured a tuple with several features, including a directed acyclic graph DAG_i, task sizes TS_{ij}, workflow arrival time AT_i, workflow deadline DT_i:

$$W_i = (DAG_i, [TS_{i1}, TS_{i2}, \ \dots TS_{ij}, \dots], AT_i, DL_i) \quad (1)$$

The above-mentioned features of a workflow are unknown before W_i arrives at the scheduling system.

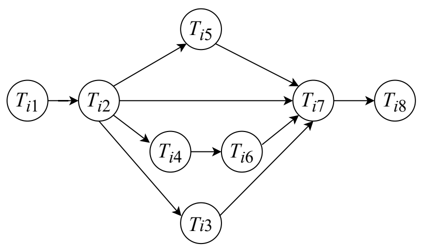

Fig. 2. A Workflow with 8 Tasks in DAG Form.

This paper uses heterogeneous VM types to solve dynamic workflow scheduling problem. Since cloud offers high level of resource elasticity, the number of each VM type is considered unlimited. Each VM instance has several inherent features, defined as:

$$V_k = (TYPE_k, CAP_k, PRICE_k) \quad (2)$$

where CAP_k represents the processing capacity of the k-th VM instance. Let $PRICE_k$ denotes the rental fee of each hour, which is related to the type of VM instance and determined solely by IaaS providers.

B. SLA Penalty

We define the time when the task T_{ij} is ready to be allocated as RT_{ij}, and denote the finish time of T_{ij} as FT_{ij}. When all the predecessor tasks of T_{ij} are finished, it becomes ready, denoted by Equation (3).

$$RT_{ij} = \max\{FT_z \mid z \in \mathrm{Precede}(T_{ij})\} \quad (3)$$

In particular, the ready time of the entry task is equal to the arrival time of this workflow, that is RT_{i1}=AT_i.

The tasks from all workflows are scheduled in the order of their ready time. Whenever *Precede(T_{ij})* are completed, T_{ij} is in the ready state. Given a ready task, the scheduling rule assigns the ready task to a specific VM for queuing, which is described in detail in Subsection III-C. The moment when a VM instance starts to process RT_{ij} is called the real start time, defined in Equation (4).

$$ST_{ij}^k = \max\left\{RT_{ij}, FT_{\mathrm{Pre}(T_{ij})}^k\right\} \quad (4)$$

where *Pre(T_{ij})* is the task processed by the same VM instance, right before T_{ij}. $FT_{Pre(T_{ij})}^k$ is the finish time of the task processed before T_{ij} on the VM instance Vk. Given the ready time and the real start time, the waiting time period of T_{ij} is defined as Equation (5).

$$WP_{ij}^{k} = ST_{ij}^{k} - RT_{ij} \tag{5}$$

Based on the inherent features of workflow and VM, the actual execution time of T_{ij} on V_k can be calculated as

$$EP_{ij}^{k} = \frac{TS_{ij}}{CAP_k} \tag{6}$$

Accordingly, the finish time of T_{ij} on V_k is calculated by Equation (7).

$$FT_{ij}^{k} = ST_{ij} + EP_{ij}^{k} \tag{7}$$

Then, the relative finish time period of a task can be defined as the sum of its waiting time and expected time.

$$RFP_{ij} = WP_{ij}^{k} + EP_{ij}^{k} \tag{8}$$

The above temporal quantities can be classified into two categories. One is the time point attributes consisting of RT_{ij}, ST_{ij}^{k}, FT_{ij}^{k}, and the nother is the time period attributes consisting of WP_{ij}^{k}, SEP_{ij}^{k}, RFP_{ij}. When all the tasks of a workflow are finished, we can calculate the makespan of this workflow, denoted as MS_i.

$$MS_i = \max\{FT_z \mid z \in W_i\} - AT_i \tag{9}$$

The deadline of the workflow is denoted as DL_i. We can calculate the shortest makespan of workflow W_i, denoted as $\min MS_i$, by processing all its tasks on the fastest VM instance without delay. However, in reality, due to the limited VM rental budget and a large number of workflows needed to be processed at the same time, it is impossible to use the fastest VM type for all workflows. Therefore, we add a relaxation coefficient ξ to define DL_i as follows.

$$DL_i = AT_i + \xi \times \min MS_i \tag{10}$$

A larger ξ implies more relaxed deadline which can be fulfilled by using relatively cheaper VMs.

Given the deadline of W_i, the deadline penalty cost of W_i is calculated by

$$DPC_i = \delta(i) \times \max\{0, AT_i + MS_i - DL_i\} \tag{11}$$

where δ is the penalty coefficient. The smaller the value of δ, the greater the tolerance for violating the deadline.

C. VM Rental Fee

The scheduling rule Φ^{sch} aims to match a ready task with a VM instance selected from a set of VM candidates. The sorting rule Φ^{sort} aims to select a pending task from the queue of a VM instance for processing. Following Φ^{sch} and Φ^{sort}, we can determine the number of rental VMs and their lease period. In reality, cloud resources are leased in units of one hour [5], so the rental fee is also calculated on a hourly basis. If the remaining lease period of V_k is less than the estimated processing time of the current and all pending tasks, the lease period of this VM will be automatically extended by one hour. Let *RVMS* denotes the set of the leased VMs, thereby the lease period LP_k of a VM instance $k \in RVMS$ can be formulated as

$$LP_k = \left\lceil \frac{FT_{T_{last}^{k}} - ST_{T_{first}^{k}}}{3600} \right\rceil \text{ where } (k \in RVMS) \tag{12}$$

where T_{first}^{k} and T_{last}^{k} are the first task and the last task processed on V_k respectively. The denominator converts the time unit from seconds to hours.

Accordingly, the VM rental fee of V_k can be calculated by Equation (13).

$$RF_k = PRICE_k \times LP_k \tag{13}$$

D. Objective

The whole process of BSDWS is described in Fig. 3. The goal of the BSDWS problem is to find a scheduling rule Φ^{sch} that can minimize the total cost of executing workflows in cloud and SLA related penalty. Particularly,

$$\min F = \sum_{i \in \mathcal{W}} DPC_i + \sum_{k \in RVMS} RF_k \tag{14}$$

where $\mathcal{W}$ is the set of all workflows to be executed, and *RVMS* is the set of leased VM instances.

IV. ALGORITHM DESIGN

This section introduces our proposed DWSGP algorithm. To design highly effective schedule rules, we introduce a new function and three new terminals for building the scheduling heuristic. Each evolved GP tree is evaluated through extensive simulations using a simulator developed recently by us. The evolution of GP trees is driven by the *crossover*, *mutation*, and *reproduction* operators.

A. Outline of DWSGP

DWSGP aims to produce a scheduling rule that allocates each ready task from the submitted workflows to available cloud resources in real time. As this paper studies a dynamic scheduling problem, we developed a cloud simulator to evaluate the scheduling rule, which is described in Subsection V-C.

The pseudo code of DWSGP is shown in Algorithm 1. Our DWSGP uses a GP tree to represent an evolved individual, i.e., a scheduling rule. Then, an initial population is established, which contains P individuals. Next, the algorithm evaluates, recombines, mutates, and reproduces individuals sequentially in each generation. In addition, DWSGP adopts an elite strategy, that is, the best three individuals in the current generation are directly retained to the next generation. When it reaches the stop criteria, DWSGP outputs the best performing individual.

B. Representation and Initialization

We first determine the representation of individual in DWSGP. Syntax tree is the most common representation in GP, where terminals are the leaves of the tree and functions are the internal nodes. Thus, the DWSGP approach represents an individual (i.e., a scheduling rule) as a GP tree. Moreover,

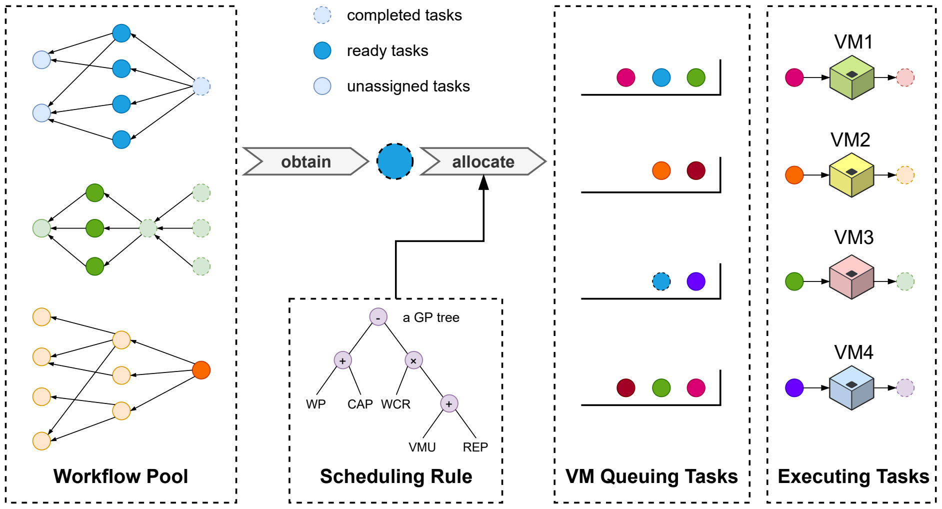

Fig. 3. Dynamic Workflow Scheduling in Cloud Computing.

Algorithm 1: DWSGP Algorithm

```
Input: A list of dynamic workflows, population size
       $P$, iteration times $G$ of DWSGP, evaluation
       times of each individual $E$, deadline relaxation
       factor $\xi$
Output: A scheduling heuristic rule
1  create the primitive set;
2  initialize the population $pop$;
3  while iteration times $\leq G$ do
4      for each scheduling rule $r$ in $pop$ do
5          while evaluation times $\leq E$ do
6              calculate the total costs of $r$;
7          end
8          evaluate the fitness;
9      end
10     tournament selection;
11     crossover, mutation, reproduction;
12     save elite individuals;
13     update $pop$;
14 end
15 Return the best scheduling rule
```

the specific composition of the primitive set is described in Table II.

In the terminal set, the terminals can be classified into four categories: the constants, task related features (TS_{ij}, WCR_i), VM related features (CAP_k, $PRICE_k$, VMU^k, $VMRT^k$), and the task temporal features (EP_{ij}^k, WP_{ij}^k, RFP_{ij}). Note that most of the existing studies [16], [17] only considered the first 6 terminals in Table II as their terminal set, also called the basic terminal set. Different from that, DWSGP adds three new parameters WCR_i, VMU^k, and $VMRT^k$ into the terminal set, defined as follows.

$$WCR_i = \frac{\text{number of completed tasks}}{\text{total number of tasks in } W_i} \tag{15}$$

TABLE II
THE TERMINAL AND FUNCTION SET FOR THE PROPOSED DWSGP APPROACH

Terminals	**Definition**
TS_{ij}	The size of the j-th task in the i-th workflow
CAP_k	Computing capacity of the k-th VM instance
$PRICE_k$	Unit price of the k-th VM instance
EP_{ij}^k	Execution time of the task T_{ij} on the k-th VM instance
WP_{ij}^k	Waiting time of the task T_{ij} on the k-th VM instance
RFP_{ij}	Relative finish time of the task T_{ij}
$Constant$	Ephemeral constant (-1, 0 or 1)
WCR_i	Completion ratio of the i-th workflow
VMU^k	Utilization ratio of the k-th VM instance
$VMRT^k$	Remain available time of the k-th VM instance
Functions	**Definition**
Add, Substract, Multiply	Basic arithmetic operations (+, -, $\times$)
Protected Division	Return 1 if the denominator is 0.(/)
If-then-else	return output1 if input else output2

$$VMU^k = \frac{\text{VM processing time}}{\text{total VM rental time } (LP_k)} \tag{16}$$

$$VMRT^k = LP_k - \text{VM processing time} - \text{VM idle time} \tag{17}$$

In particular, WCR_i represents the completion degree of a workflow, VMU^k reflects the utilization efficiency of the VM. $VMRT^k$ is the remaining available time of a VM instance. In this case, if the execution time of a ready task is larger than $VMRT^k$, the rental time of this VM will be extended. The above factors may affect the scheduling results which are not considered in existing studies. Further analysis will be presented in Section V-E.

As for the function set, we choose the basic arithmetic operators, such as Add, Subtract, Multiply, and Protected Division. In addition, an If-then-else function node is added, which is not considered in previous research works.

C. Fitness

In DWSGP, we use a cloud simulator to evaluate evolved GP trees. For each simulation, the input of the cloud simulator contains mixed random workflows and a scheduling rule. The simulator obtains ready tasks from the simulation environment in real-time, and then allocates all ready tasks until all workflows are completed. In particular, the scheduling rule aims to calculate the priority values of all available VM instances for the ready task. Then, the ready task is allocated to the VM instance with the highest priority value. After processing all workflows, the cloud simulator outputs the total cost with respect to SLA penalties and VM rental fees as the fitness of an individual.

D. Crossover, Mutation, Reproduction

$Crossover$ is realized through random exchange between two parent individuals. In particular, the DWSGP algorithm randomly selects a crossover point in each parent tree, copy the tree of one parent as $\mathrm{A}_{\mathrm{copy}}$, and copy the subtree of the second parent starting from the crossover point as $\mathrm{B}_{\mathrm{subtree\_copy}}$. Then, subtree under the crossover point of $\mathrm{A}_{\mathrm{copy}}$ is replaced with $\mathrm{B}_{\mathrm{subtree\_copy}}$. Similarly, the remaining two parts $\mathrm{B}_{\mathrm{copy}}$ and $\mathrm{A}_{\mathrm{subtree\_copy}}$ are reorganized into another offspring.

$Mutation$ is achieved through a random partial change of a parent individual. A mutation point is randomly selected in a parent tree. Then, the DWSGP algorithm randomly generate a subtree. The subtree of the parent tree with the mutation point as the root node is replaced with this randomly generated subtree.

$Reproduction$ means that individuals are copied directly into the next generation.

V. EXPERIMENTS

In this section, we first introduce the datasets and the setup of the experiments. Then, we compare the proposed DWSGP with several existing heuristics and conventional GP approaches to verify the effectiveness of DWSGP. After that, the performance of our proposed DWSGP is evaluated under different deadline relaxation factors.

A. Datasets

This paper considers a cloud center equipped with 6 different VM types. Table III provides a summary of VM instances collected in February 2021 from Amazon EC2[1]. A larger vCPU indicates a faster processing speed as well as a higher hourly rental price.

Our cloud simulator supports a workflow set consists of 4 classic patterns[2], namely CyberShake, Inspiral, Montage, and SIPHT. In this paper, we set up three different sizes for each pattern (e.g., small, medium, and large), so the workflow pool has a total of 12 different workflows, as shown in Table IV. Every time the simulation is performed, 50 workflows are randomly sampled from the 12 workflow patterns. Besides, the arrival time of workflows follows a Poisson distribution with $\lambda = 0.01$ [4], and the penalty coefficient δ in Equation (11) is set to be \$0.24 per hour [24]. As for the deadline relaxation coefficient ξ in Equation (10), we increase it gradually from 6 to 42 with a step of 6. A larger ξ represents a looser deadline and the simulator is more likely to choose the slower but cheaper VM instances. In particular, when ξ=1, the deadline interval is equivalent to the total execution time of all workflows running on the fastest VM, while ξ=24 encourages the SaaS provider to use the slowest VM.

[1]https://aws.amazon.com/ec2/pricing/on-demand/

[2]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowHub

TABLE III
CONFIGURATION INFORMATION OF DIFFERENT VMS BASED ON AMAZON EC2

Type	vCPU	ECU	Memory (GiB)	Cost (\$/hr)
m5.large	2	10	8	0.096
m5.xlarge	4	16	16	0.192
m5.2xlarge	8	37	32	0.384
m5.4xlarge	16	70	64	0.768
m5.8xlarge	32	128	128	1.536
m5.12xlarge	48	168	192	2.304

TABLE IV
NUMBER OF TASKS IN DIFFERENT WORKFLOWS

Size	CyberShake	Inspiral	Montage	SIPHT
Small	30	30	25	30
Medium	50	50	50	50
Large	100	100	100	100

B. Parameter Settings

Following [25], we set the population size to 1024, the number of generations to 50, and the tournament size to 4. The crossover, mutation and reproduction rate are 85%, 10% and 5%, respectively. Besides, we set the initial depth of a tree to be in the range from 2 to 6, and the maximum depth of a tree to be 7. The evaluation frequency of each individual is set to 3. All the experiments are conducted on computers with Intel Core i7-8700 3.2GHz CPUs, 6 cores and 16GB RAM.

C. Baselines and Comparison

To evaluate the performance of our proposed DWSGP, we compare it with six workflow scheduling heuristics [9], [16], [26], [27] listed below. Moreover, the approaches in Table V are added for further comparison. All of these baselines are implemented on our Cloud Simulator. Table V shows the relationship between SGP, NFGP, NTGP and DWSGP.

1) HEFT, known as Heterogeneous Earliest Finish Time.
2) FCFS, namely First Come First Scheduling, allocates the first ready task to the first idle VM.
3) CHP, a greedy algorithm that always selects the cheapest available VM to minimize the budget.
4) SGP, standard GP with normal terminal and function set
5) NFGP, namely GP with a New Function, allows SGP to use the if-then-else function node to build GP trees.

6) NTGP, namely GP with New Terminals. This method can use the three new terminal nodes WCR_i, VMU^k, and $VMRT^k$ to evolve scheduling rules.

TABLE V
COMPOSITION OF THE GP-BASED APPROACHES

Baseline Name	**New Terminal** (WCR_i, VMU^k, **and** $VMRT^k$)	**New Function** ($if-then-else$)
SGP	✗	✗
NFGP	✗	✓
NTGP	✓	✗
DWSGP	✓	✓

D. Simulation Results

Table VI records the average value of total costs of all baselines after 30 independent runs, where the total costs contain the VM rental fees and the SLA penalties. A smaller value represents a better performance. In Table VI, a total of 7 approaches are performed under three different deadline relaxation coefficients. As shown in Table VI, DWSGP consistently achieved the lowest total cost under different ξ comparing with other baselines. In other words, the proposed DWSGP has the best performance in different settings.

TABLE VI
TOTAL COSTS UNDER DIFFERENT ξ

Algorithm	$\xi = 1$	$\xi = 12$	$\xi = 24$
HEFT	158.29 ± 2.95	139.73 ± 2.72	69.39 ± 2.82
FCFS	218.28 ± 12.89	178.53 ± 18.26	169.14 ± 18.10
CHP	717.62 ± 73.00	726.60 ± 48.39	724.22 ± 70.15
SGP	109.85 ± 7.25	68.12 ± 3.14	62.74 ± 1.01
NFGP	107.77 ± 3.23	72.79 ± 4.75	61.47 ± 1.42
NTGP	105.11 ± 2.23	66.40 ± 2.57	59.16 ± 0.89
DWSGP	$\mathbf{103.67 \pm 1.67}$	$\mathbf{65.62 \pm 1.84}$	$\mathbf{58.33 \pm 1.20}$

The total cost becomes smaller as ξ increases for almost all baseline approaches. This is because relaxing deadline interval will increase the probability of choosing cheaper resources. Besides, we see that the HEFT, FCFS, and CHP methods consistently perform worse than the GP-based approaches. HEFT always selects the available VM with minimum earliest finish time, while FCFS chooses the VM that has shortest queuing time regardless of execution time. Thus, the performance of HEFT is better than FCFS, but not better than SGP. When the deadline is relaxed, the difference between HEFT and SGP becomes smaller. Comparing SGP and NFGP methods, the gap between them is small across different deadline factors, indicating that adding if-then-else is beneficial but does not have a significant effect on SGP. Compared with SGP, the result of NTGP is smaller in three parameter settings, so NTGP performs better than SGP. Among all the methods, NTGP is performed secondary, which is just 1.4%, 1.2%, 1.4% worse than our DWSGP method under the three parameter settings, respectively.

E. Further Analysis

The iteration curves of our DWSGP approach with 8 different deadline relaxation factors are described in Fig. 4. The horizontal axis is the generation index, and the vertical axis is the total costs (i.e., fitness). Each experiment undergoes 30 independent evaluations to obtain the mean curve of the total costs.

We can see that all the 8 curves have converged. The iterative curve moves downward as ξ increases. A larger ξ means a looser deadline. In that case, more slower but cheaper VM instances are likely to be selected to handle tasks. In Fig. 4, the difference in total costs between the first generation and the last generation decreases as ξ increases. When ξ is large enough, the scheduling rules obtained across all generations tend to produce similar total costs. Since it is highly unlikely to violate the SLA penalty, the focus of the BSDWS problem with large ξ is simply to minimize the VM rental fees.

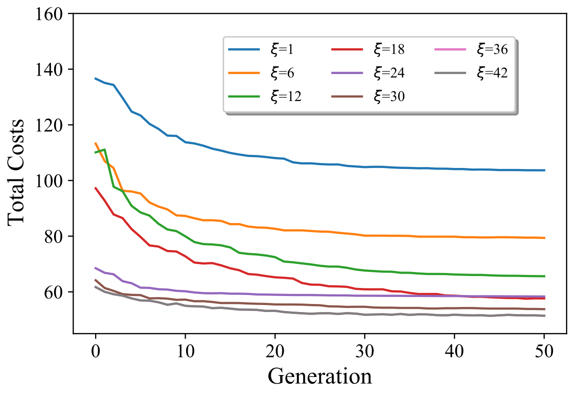

Fig. 4. DWSGP Performance under Different Deadline Settings.

Fig. 5 shows the iteration curves of SGP, NFGP, NTGP, and DWSGP under three different parameter settings. When ξ=1 and ξ=24, the convergence speed of SGP is relatively slow. In addition, the approaches with new terminal nodes outperformed those without, e.g., NTGP beats SGP, and DWSGP beats NFGP. When ξ=12, SGP, NFGP, and NTGP have similar convergence speed and final performance, but NFGP clearly converged slower than other methods. We also find that the effect of if-then-else node is not particularly significant. On the contrary, our new introduced terminals (WCR_i, VMU^k, $VMRT^k$) are effective at all times. In summary, the proposed DWSGP achieved consistently superior performance than other baseline algorithms in all our experiments.

VI. CONCLUSIONS

This paper proposes a new GP approach to automatically evolve heuristics for the BSDWS problem.We took into account heterogeneous workflows in simulation, and introduced three new terminal nodes with empirically verified importance for building schedule rules. We also considered the deadline violation penalty as an optimization objective the first time in literature. Experiments showed that our approach can evolve

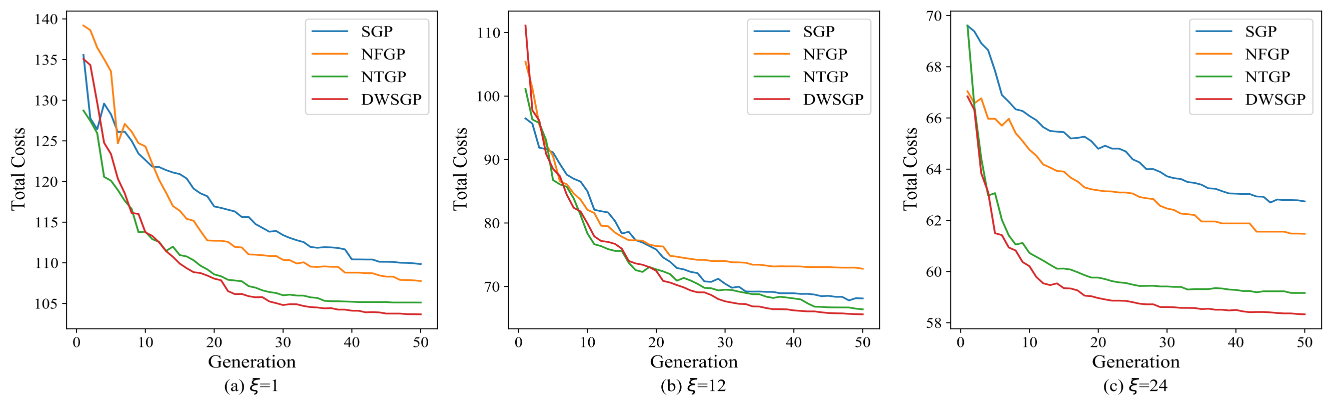

Fig. 5. Performance of Four GP-based Algorithms.

effective heuristics to significantly reduce the total cost than other competing algorithms. Compared with other algorithms, the proposed DWSGP approach consistently achieves the best performance under different settings of ξ. Moreover, the BSDWS problem with a lower SLA constraints (i.e., a larger ξ) leads to lower VM rental fees. In the future, we will study how to improve the computation efficiency of the DWSGP algorithm. This can be potentially realized by adopting a surrogate model.

References

[1] D. J. Hejji, A. B. Nassif, Q. Nasir, and M. AbuTalib, "Systematic literature review: Metaheuristics-based approach for workflow scheduling in cloud," in *2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*, 2020, pp. 1–5.

[2] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[3] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1344–1357, 2016.

[4] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, and N. Najjari, "Online multi-workflow scheduling under uncertain task execution time in iaas clouds," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2019.

[5] H. R. Faragardi, M. R. Saleh Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, "Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1239–1254, 2020.

[6] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget and deadline aware e-science workflow scheduling in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 29–44, 2019.

[7] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11.

[8] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, and J. Wen, "Deadline-constrained cost optimization approaches for workflow scheduling in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3401–3412, 2017.

[9] H. Topcuoglu, S. Hariri, and Min-You Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[11] B. P. Rimal and M. Maier, "Workflow scheduling in multi-tenant cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 290–304, 2017.

[10] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 400–407.

[12] A. C. Zhou and B. He, "Transformation-based monetary costoptimizations for workflows in the cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 85–98, 2014.

[13] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, 2017.

[14] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, 2013.

[15] Y. Liu, Y. Mei, M. Zhang, and Z. Zhang, "A predictive-reactive approach with genetic programming and cooperative coevolution for the uncertain capacitated arc routing problem," *Evolutionary computation*, vol. 28, no. 2, pp. 289–316, 2020.

[16] Y. Yu, Y. Feng, H. Ma, A. Chen, and C. Wang, "Achieving flexible scheduling of heterogeneous workflows in cloud through a genetic programming based approach," in *CEC2019*, 2019, pp. 3102–3109.

[17] K.-R. Escott, H. Ma, and G. Chen, "Genetic programming based hyper heuristic approach for dynamic workflow scheduling in the cloud," in *International Conference on Database and Expert Systems Applications*. Springer, 2020, pp. 76–90.

[18] L. Zeng, B. Veeravalli, and X. Li, "Saba: A security-aware and budget-aware workflow scheduling strategy in clouds," *Journal of parallel and Distributed computing*, vol. 75, pp. 141–151, 2015.

[19] L.-j. Jin, F. Casati, M. Sayal, and M.-C. Shan, "Load balancing in distributed workflow management system," in *Proceedings of the 2001 ACM symposium on Applied computing*, 2001, pp. 522–530.

[20] Z. Chen, K. Du, Z. Zhan, and J. Zhang, "Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm," in *CEC2015*, 2015, pp. 708–714.

[21] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, and D. Yuan, "A genetic algorithm based data replica placement strategy for scientific applications in clouds," *IEEE Transactions on Services Computing*, vol. 11, no. 4, pp. 727–739, 2018.

[22] W. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 1, pp. 29–43, 2009.

[23] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated design of production scheduling heuristics: A review," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, 2015.

[24] C.-H. Youn, M. Chen, and P. Dazzi, *Cloud Broker and Cloudlet for Workflow Scheduling*. Springer, 2017.

[25] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[26] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[27] J. O. F. Comellas, Í. G. Presa, and J. G. Fernández, "Sla-driven elastic cloud hosting provider," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 111–118.