Evolving Scheduling Heuristics for Energy-Efficient Dynamic Workflow Scheduling in Cloud via Genetic Programming Hyper-Heuristics

Zaixing Sun[1,2], Fangfang Zhang[2], Yi Mei[2], Hejiao Huang[1], Chonglin Gu[1(✉)], Bin Qian[3], and Mengjie Zhang[2]

[1] Harbin Institute of Technology (Shenzhen), Shenzhen 518000, China
guchonglin@hit.edu.cn
[2] Victoria University of Wellington, Wellington 6140, New Zealand
[3] Kunming University of Science and Technology, Kunming 650500, China

Abstract. With the rapid development of cloud computing, the issue of how to reduce energy consumption has attracted a great deal of attention. Especially for dynamic workflow scheduling, dependency constraints between tasks and high quality of service requirements, such as real-time requirements and deadline constraints, make it very challenging. This paper focuses on the energy-efficient scheduling problem, which jointly considers the impact of finer-grained tasks with CPU and memory configurations on energy consumption. A dynamic workflow scheduling simulator is developed to mimic the scheduling process in real-world scenarios. Then, we propose a Cooperative Coevolution Genetic Programming to learn heuristics for both the task selection decision and the instance selection decision, using the simulator for heuristic evaluation. The scheduling heuristics obtained by Cooperative Coevolution Genetic Programming evolution can then be used to make real-time decisions in dynamic environments. The simulation results show that the proposed method has managed to obtain better scheduling heuristics than the baseline methods in terms of energy consumption and resource utilization.

Keywords: Cloud Computing · Dynamic Workflow Scheduling · Genetic Programming Hyper-heuristics

1 Introduction

With the rapid growth of cloud computing, the energy consumption of data centers is expected to rise significantly, from 200 terawatt hours (TWh) in 2016 to a staggering 2967 TWh by 2030 [1]. This surge in energy consumption not only increases data center operating costs, but also exacerbates environmental damage. The workflow is a popular paradigm for the execution of scientific and industrial applications in cloud and consists of a set of interdependent tasks, where a task may be connected to one or more

D.-S. Huang et al. (Eds.): ICIC 2024, LNCS 14862, pp. 169–182, 2024.
https://doi.org/10.1007/978-981-97-5578-3_14

tasks through directed edges. The directed edge usually means execution constraints or data transfer dependencies between tasks. Efficient workflow management, specifically focusing on energy-efficient dynamic workflow scheduling, is essential for orchestrating tasks to optimize performance, reduce costs, and minimize energy consumption in cloud computing.

Workflow scheduling problems have been extensively studied and various heuristic algorithms have been proposed. Rodriguez et al. [2] studied dynamic workflow scheduling in cloud. They considered a model in which tasks executed in containers, which in turn were deployed on virtual machines (VMs). Ye et al. [3] studied the joint off-line batch workflows and online stream workflows scheduling for minimizing the cost and improving resource utilization in cloud container services. Although this model takes advantage of the faster startup of the container than VM, it ignores the elasticity of the container. Havet et al. [4] reduced energy consumption in clusters by using generational garbage collection principles, while striving to meet deadline constraints for all requests as closely as possible. Hu et al. [5] proposed adjustments to task scheduling decisions that balance energy consumption with task execution times in real-time environments. Stavrinides et al. [6] and Fan et al. [7] studied dynamic workflow scheduling with deadline-constrained and aimed to reduce energy consumption while ensuring service quality constraints. For system models in [6] and [7], they considered not only task-to-VM deployment, but also VM-to-host deployment. Their energy consumption in [6] and [7] was related to the frequency of the VMs and the execution time of the tasks. Sun et al. [8] focused on the real-time energy-saving multi-workflow scheduling on container cloud, which considers resource quantity and affinity relationship between containers and physical machines. However, these studies are limited to the impact of the amount of computation of the task or the frequency of the VM on the energy consumption, and they do not simultaneously consider the types of tasks with CPU and memory configurations to execute the tasks in parallel on the VM. There are also no existing studies that consider the impact of having both CPU and memory configurations on energy consumption, which is a more fine-grained study.

Hyper-heuristic [9, 10] is a promising approach for automatically selecting or generating heuristics to solve hard computational search problems. The goal of this approach is to explore the "heuristic search space" of the problems instead of the solution search space in the cases of heuristics and meta-heuristics. Genetic programming as a hyper-heuristic method was investigated in [11] to evolve scheduling rules. The purpose of using GPHH is to improve the generalization of scheduling rules by generating new heuristics using existing heuristics. It has successfully solved many problems in various fields such as job shop scheduling [12] and workflow scheduling[13, 14]. Traditional workflow scheduling in cloud usually involves two decisions [15, 16]: the task prioritization and the resource selection. Based on these two decisions, two high-level heuristics are derived to select a ready task and to select a resource to execute the ready task, respectively. Cooperative Coevolution Genetic Programming (CCGP) [17, 18] is a mechanism for effectively evolving coadapted subcomponents and efficiently learning high-quality coadapted sub-heuristics.

This paper focuses on more fine-grained CPU and memory configuration of tasks and establishes an energy-efficient dynamic workflow scheduling (E2DWS) model. When

the CPU and memory configuration of tasks are satisfied, multiple tasks can be executed on the same resource at the same time, which affects the resource utilization rate and further affects the energy consumption. E2DWS inherits many challenges of workflow scheduling in traditional cloud, e.g., deadline constraints; determining the scheduling sequence of tasks. Since we consider the CPU and memory configuration of tasks, this leads to the following extra challenges in E2DWS: how to ensure that concurrent tasks do not exceed the VM's configuration when scheduling the workflow; how to execute tasks in parallel as much as possible to save energy. To address these challenges, this paper proposes CCGP to evolve scheduling heuristics for E2DWS. The main contributions of this work are summarized as follows.

- An energy-efficient dynamic workflow scheduling model is established, which jointly considers more fine-grained CPU and memory configuration of tasks that affect the energy consumption.
- We develop a dynamic workflow scheduling simulator to mimic the scheduling process in real-world scenarios. Based on the simulator, we propose a CCGP method for evolving the allocation rules for two levels simultaneously.
- We verify the effectiveness of the proposed CCGP method by comprehensive empirical study based on real-world data traces.

2 Cloud Workflow Scheduling Model

Let $\mathcal{R}$ be the set of heterogeneous computational resources. Each resource r is associated with a type $\theta(r) \in \Theta$, the size of memory $\Gamma^m(r)$ and the size of CPU $\Gamma^c(r)$. Let B_{WAN} be the bandwidth of the wide area network (WAN) and L_{WAN} be the latency of the WAN. Static power $p^s(r)$ includes base power $p^b(r)$, CPU-based idle power $p^{ic}(r)$ and memory-based idle power $p^{im}(r)$, where $p^s(r) = p^b(r)+p^{ic}(r)+p^{im}(r)$. Dynamic power is defined as the difference between full power and idle power, including CPU-based power $p^{dc}(r) = p^{fc}(r)-p^{ic}(r)$ and memory-based power $p^{dm}(r) = p^{fm}(r)-p^{im}(r)$. Let $\mathcal{G}$ be a set of workflow applications. Each workflow $g \in \mathcal{G}$ contains a set of tasks $A(g)$. Each task $\alpha \in A(g)$ is represented by a tuple $\{\tau^e(\alpha), \mu^c(\alpha), \mu^m(\alpha), \mathrm{Pre}(\alpha), \mathrm{Suc}(\alpha)\}$, where $\tau^e(\alpha)$ is the reference execution time, $\mu^c(\alpha)$ is computational resource requirement, $\mu^m(\alpha)$ is memory resource requirement, $\mathrm{Pre}(\alpha) \subseteq A(g)$ is a set of predecessor tasks, and $\mathrm{Suc}(\alpha) \subseteq A(g)$ is a set of successor tasks. $d(\alpha_1, \alpha_2)$ is the data required to be transferred from task α_1 to its successor task $\alpha_2 \in \mathrm{Suc}(\alpha_1)$. The arrival time of the workflow g is $\rho^a(g)$, and its deadline is $\rho^d(g)$.

We formulate the dynamic workflow scheduling as an optimization problem aiming to minimize the energy consumption and to maximize resource utilization, which subject to the following constraints: No task can be executed until all its predecessor tasks have been completed; Each resource can deploy and run multiple tasks simultaneously, but the sum of the CPU and memory capacities of the tasks allocated on the resource cannot exceed the corresponding resource's capacity at any moment. We assume scheduling as a discrete-time control problem as standard in previous work [19, 20], i.e., each decision round as a time slot. At the beginning of each time slot, the Cloud Broker makes task scheduling and resource deployment and scaling decisions. We divide the timeline into equal duration time slots $T = \{0,1,2,\cdots,t,\cdots,\}$, where t is the t th time slot. Let

$x^s(\alpha)$ and $x^f(\alpha)$ be the execution start time and execution finish time of the task α. Let $y(\alpha) \in \mathcal{R}$ be assigned resource of the task α in the schedule. Let $v^c(\alpha, t)$ and $v^m(\alpha, t)$ be the sizes of the CPU and memory allocated to the task α at time t.

$$min \quad E = \sum\nolimits_{t \in T} \sum\nolimits_{r \in \mathcal{R}} P(r, t) \tag{1}$$

$$s.t.: \quad x^s(\alpha) \geq \rho^a(g), \forall g \in \mathcal{G}, \alpha \in A(g), \tag{2}$$

$$x^f(\alpha) \leq \rho^d(g), \forall g \in \mathcal{G}, \alpha \in A(g), \tag{3}$$

$$\begin{cases} v^c(\alpha, t) = \mu^c(\alpha), \forall t \in T, \\ v^m(\alpha, t) = \mu^m(\alpha), \forall t \in T, \end{cases} \tag{4}$$

$$\begin{cases} \sum_{\forall y(\alpha)=r} v^c(\alpha, t) \leq \Gamma^c(r), \forall r \in \mathcal{R}, \forall t \in T, \\ \sum_{\forall y(\alpha)=r} v^m(\alpha, t) \leq \Gamma^m(r), \forall r \in \mathcal{R}, \forall t \in T, \end{cases} \tag{5}$$

$$x^c(\alpha_1, \alpha_2) = \begin{cases} 0, & y(\alpha_1) = y(\alpha_2), \\ L_{WAN} + \frac{d(\alpha_1, \alpha_2)}{B_{WAN}}, & \text{otherwise}, \end{cases} \tag{6}$$

$$x^s(\alpha_1) + \tau^e(\alpha) + x^c(\alpha_1, \alpha_2) \leq x^s(\alpha_2), \forall \alpha_2 \in Suc(\alpha_1), \tag{7}$$

$$P(r, t) = p^s(r) + p^{dc}(r) \times U^c(r, t) + p^{dm}(r) \times U^m(r, t), \tag{8}$$

where $\forall g \in \mathcal{G}$, $\forall \alpha, \alpha_1, \alpha_2 \in A(g)$, $\forall r \in \mathcal{R}$ and $\forall t \in T$. Equation (1) is the total energy consumption. Constraint (2) ensures that the workflow cannot be executed until it is submitted. Constraint (3) ensures that the workflow is finished before its deadline. Constraint (4) indicates that the configuration needs to satisfy the requirements of the task. Constraint (5) indicates that the sum of the CPU and memory capacities of the tasks allocated on the resource cannot exceed the corresponding resource's capacity at time t. Equation (6) is the communication time between each task α_1 and its successor task α_2. Constraint (7) is the precedence constraint in a workflow, which indicates that the task cannot be executed until all of its immediate predecessors have finished and it has received all of the output data from its immediate predecessors. Equation (8) is the power of different resources at time t.

3 Discrete Event-Driven Simulation Process

For fitness evaluation, we develop a discrete event-driven simulation process with the four rules, which maintains an event queue $\mathcal{E}$ sorted with trigger time. Each event $\mathcal{E}_\urcorner$ is associated with the event name $\mathcal{E}_\urcorner^{\mathcal{N}}$, trigger time $\mathcal{E}_\urcorner^{\mathcal{T}}$, workflow involved $\mathcal{E}_\urcorner^{\mathcal{W}}$, task involved $\mathcal{E}_\urcorner^{\mathcal{A}}$, and service instance involved $\mathcal{E}_\urcorner^{\mathcal{S}}$. According to the event name, we introduce *when* each event is triggered and *what* will happen when the event is triggered:

- WorkflowSubmitted. *When*: the arrival time of a workflow. *What*: set the tasks with no immediate predecessors to ready state. If there are multiple workflow submissions or multiple tasks that need to be set to ready state, the *task selection rule* is adopted to calculate the priority of each task, and the tasks are added to the TaskReady event queue in ascending order of this priority.
- TaskReady. *When*: after a workflow has been submitted or all of a task's immediate predecessors have been assigned and started to execute. *What*: select an instance to execute the task according to the instance selection rule. Once the instance that will execute the task has been determined, we can determine the start execution time of the task from the terminals associated with that instance. Add the task to the wait queue of the instance that will execute it.
- StartExecuteTask. *When*: after the ready task has received data from all predecessors and the assigned instance is active. *What*: remove the task from the wait queue and add it to the run queue.
- CompleteExecuteTask. *When*: after the execution of task is finished. *What*: Remove the task from the run queue. Trigger the CompleteExecuteWorkflow event if this task is the last finished task in a workflow.
- CompleteExecuteWorkflow. *When*: after all the tasks in a workflow have been finished. *What*: record that the workflow has been scheduled.

The state of the simulated system, such as the system time, is updated as each event is triggered, and an event can generate and/or trigger other events. Overall, the simulation process is scheduled sequentially according to the event queue. The scheduling heuristic makes a decision on each decision point to make the scheduling process continue.

4 The Proposed Algorithm

4.1 Overview

We propose the CCGP approach to automatically generate the following 2 rules to make decisions:

- **Task selection rule** (Rule 1): to select the next ready task to be executed. If there are multiple tasks ready at the same time, the rule is used to determine the sequence of tasks.
- **Instance selection rule** (Rule 2): to select a resource in the cloud to execute the selected task. First, eliminate the instances that cannot meet the minimum requirements of CPU or Memory of the task, and the remaining instances are available. Second, it filters out the instances that cannot meet the sub-deadline to execute the task. If no instance remains, then the instance in the cloud with the earliest finish time is selected to execute the task from all available instances. Otherwise, the instance selection rule is used to select the instance. In cases where multiple instances have the same priority, prioritize selecting an existing instance that does not need to be created.

The training process is performed offline and the trained rules are deployed to make online scheduling decisions. Figure 1 shows an overview of the training process of

CCGP. CCGP evolves two populations of rules. The evolutionary mechanisms of CCGP are presented separately as follows.

- **Populations initialization**. A number of individuals for each population are initialized by random selecting and combining the terminals and functions with the ramped-half-and-half method [12].
- **Individual evaluation**. Given a training instance, we can evaluate the fitness of an individual (i.e., a combination of the two rules) by applying it to dynamic workflow scheduling simulator. In each generation, the elitism individuals of two populations are selected to cooperate with the rules from the other population and to evaluate the corresponding individuals. At the first generation, we randomly select one representative rule for evaluation. After fitness evaluation, each individual has a fitness that represents its quality.
- **Parent selection**. If the *stopping criterion* is met, the best individual so far is considered as the best evolved scheduling heuristic for E2DWS. Otherwise, we use the tournament selection to select parents for generating offspring.
- **Evolution**. Evolution employs *reproduction*, *crossover*, and *mutation*. *Reproduction* copies top performers directly to the next generation. *Crossover* involves selecting and swapping sub-trees between two parents to form new individuals. *Mutation* replaces a randomly selected subtree in a parent with a new one, ensuring variety in the population.

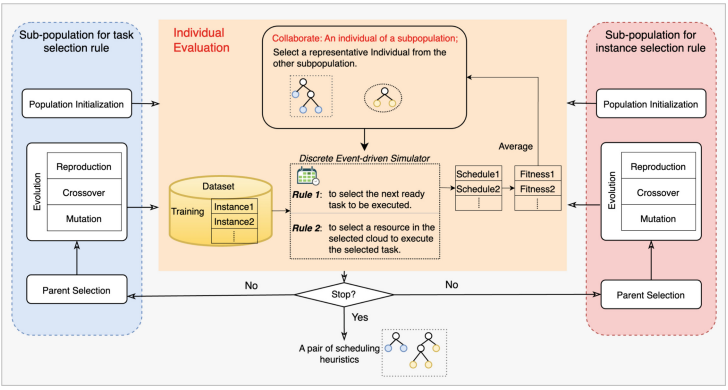

Fig. 1. Overview of CCGP.

4.2 Representation, Terminal Set, and Function Set

To evolve a scheduling heuristic with two rules for E2DWS, an individual is designed with a tree which represent task selection rule or instance selection rule. We design 21 terminals based on the scheduling process, which indicate the characteristics related to workflows, tasks, instances, and cloud, to describe the state of the cloud system. Note that the terminal set is different for each rule, but the function set is the same.

Table 1 shows the terminal and function sets of CCGP. We use the symbol "✓" to mark the terminators that compose the corresponding rule. SD is the sub-deadline of a task, which is set based on the latest finish time, as shown in Eqs. (9–10). ε is a random factor within the range [0.95,1]. The smaller the ε, the more urgent the tasks in a workflow and the smaller the sub-deadline. Following [21], we set $\varepsilon = 0.95$. We calculate the upward rank of a task following the HEFT [22], which is the length of the critical path from α_1 to the exit task, as shown in Eq. (11).

$$\iota(\alpha_1) = \begin{cases} \rho(g), & \text{if } Suc(\alpha_1) = \varnothing, \\ \max\limits_{\alpha_2 \in Suc(\alpha_1)} \left\{ \iota(\alpha_2) - \tau^e(\alpha_2) - \frac{d(\alpha_1,\alpha_2)}{B_{WAN}} \right\}, & otherwise. \end{cases} \tag{9}$$

$$SD(\alpha) = \begin{cases} \iota(\alpha), & \text{if } Suc(\alpha) = \varnothing, \\ \varepsilon \times \iota(\alpha), & otherwise. \end{cases} \tag{10}$$

$$UR(\alpha_1) = \begin{cases} \tau^e(\alpha_1), \text{ if } Suc(\alpha_1)\varnothing, \\ \tau^e(\alpha_1) + \max\limits_{\alpha_2 \in Suc(\alpha_1)} \left\{ UR(\alpha_2) + \frac{d(\alpha_1,\alpha_2)}{B_{WAN}} \right\}, otherwise. \end{cases} \tag{11}$$

Table 1. Terminal and Function sets of CCGP.

No	Nation	Description	Task	Instance
1	CA	The computational resource requirement of a task	✓	
2	MA	The memory resource requirement of a task	✓	
3	NPK	The number of direct predecessors for a task	✓	
4	NSK	The number of direct successors for a task	✓	
5	SD	The sub-deadline of a task	✓	
6	UR	The upward rank of a task	✓	
7	CT	The communication time for a task	✓	
8	NKQ	The number of tasks in ready queue	✓	
9	NRK	The number of unscheduled tasks in a workflow	✓	
10	TETQ	The total execution time of tasks in ready queue	✓	
11	TETRK	The total execution time of unscheduled tasks in a workflow	✓	
12	ET	The execution time of a task	✓	✓

(*continued*)

Table 1. (*continued*)

No	Nation	Description	Task	Instance
13	CI	The CPU size of an instance		✓
14	MI	The memory size of an instance		✓
15	STP	The static power of an instance		✓
16	AAT	The actual available time of a instance		✓
17	CTI	The communication time of a task on an instance		✓
18	CRCPU	The current remaining CPU of an instance		✓
19	CRMEM	The current remaining memory of an instance		✓
20	ST	The slack time of task, which is (AAT-SD)		✓
21	CON	Constant 0		✓
Function set		+, −, *, protected /, Max, Min	✓	✓

5 Performance Evaluation and Results

5.1 Simulation Environment Setup

To the best of our knowledge, there has been no such and similar study in the current literature. To evaluate our proposed algorithm, three benchmark algorithms are employed and operated under the same running environment with our algorithm: NCGP, Random and Binpacking. NCGP is a variant of CCGP which does not evolve by co-evolution. NCGP has only one population and each of its individuals consists directly of two rules. Comparing with NCGP, we can find the effect of co-evolution. Practically, NCGP, similar to the algorithms proposed in [13] and [14], is based on the traditional GP framework. The algorithms proposed in [13] and [14] need to consider the extra features of our problem by extending the terminal set so that they become NCGP in our comparison algorithm. In other words, the comparison algorithm NCGP is essentially the application of these algorithms to our problem. Random is one of the default scheduling strategy on the Docker Swarm (version v1.12). Random is to choose randomly from candidate tasks or instances when making decisions. Binpacking is based on classical HEFT and Binpacking strategy (one of the default scheduling strategy on the Docker Swarm (version v1.12)): task selection is based on upward rank value; instance selection is based on the utilization of instances to make the resources as busy as possible in order to save resources.

In simulation experiment, we assume that the customer can lease an unlimited number of resources. We use 5 representative VM types from low configuration to high

Table 2. Configurations of Cloud Severs.

Type	m1.small	m3.medium	m3.large	m3.xlarge	m3.2xlarge
ECU	1	3	6.5	13	26
Memory (GB)	1.7	3.75	7.5	15	30
CPU-based idle power	0.72	0.87	1.73	3.47	6.94
CPU-based full power	5.76	6.97	13.94	27.87	55.74
Memory-based idle power	0.34	0.75	1.5	3	6
Memory-based full power	1.02	2.25	4.5	9	18
Base power	1.2	1.4	2.9	5.8	11.5

configuration. The VM configurations and their processing capacity are based on current Amazon EC2 platform[1] and Teads Engineering[2] (which provide a Carbon footprint estimator for AWS instances), as presented in Table 2. As for the booting time of VM, the cold startup time is set to 55.9s [15]. The latency of the WAN delay L_{WAN} is set to 0.0005s. The Bandwidth B_{WAN} is set to 2GB/s. We consider the real cluster data released by cloud computing vendors. The cluster data released by Alibaba in 2018[3] contains the DAG information of workflow that obtains the interdependence between tasks, which is very consistent with the purpose of this study. Therefore, we select Alibaba cluster-trace-v2018 in this study. We extract 1200 DAG structures from the trace, where each DAG has at least 10 nodes. Although these public data pro-vide the DAG structure and the execution time of each task, we need to construct data applicable to our problems based on the known information. The input and out-put data of each task is assigned by a uniform discrete distribution between 500 and 5000 MB. Each DAG's deadline factor is randomly selected from the set {0.8,1.0,1.5,1.8}. Workflow arrival is modeled using the Poisson distribution with an arrive rate λ where the interarrival time is exponentially distributed with $1/\lambda$, where $\lambda \in \{0.1,0.2,0.5,0.8,1.0\}$. We partition the 1200 DAGs into 5 datasets, {200, 100, 200, 300, 400}. We randomly select 10% the workflows from these 5 datasets to simulate dynamic scenarios.

We implement the simulation and algorithms in Python 3.10 and deploy it on a computer with a 3.80 GHz Intel Core i7-10700K processor and 15.3 GB of memory. The parameters of the CCGP are shown in Table 3. For all the compared algorithms, 30 independent runs are done for each scenario and the evolved scheduling heuristics are tested on 30 unseen instances. The average objective value across the 30 test instances is reported as the test performance of the rule, which can be a good approximation of the true performance of the rules. Along with 30 independent runs, Friedman's test and Wilcoxon rank-sum test with a significance level of 0.05 are used to examine the performance of algorithms. The "−", "+", and "≈" indicate that the corresponding result

[1] https://aws.amazon.com/cn/ec2/instance-types/

[2] https://engineering.teads.com/sustainability/carbon-footprint-estimator-for-aws-instances/

[3] https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018/trace_2018.md.

is significantly better than, worse than, or similar to its counterpart. An algorithm will be compared with the algorithm(s) before it one by one.

Table 3. The parameter settings of CCGP and NCGP

Parameter	Value
Population size	50 for each subpopulation/100 for NCGP
Number of generations	50
Method for initializing population	Ramped-half-and-half
Initial minimum/maximum depth	2/7
Elitism	5
Maximal depth	8
Crossover rate	0.8
Mutation rate	0.15
Reproduction rate	0.05
Parent selection	Tournament selection with size 7
Terminal/non-terminal selection rate	10%/90%

5.2 Results

Based on the our model, we also can get the total resource utilization:

$$U = \frac{\sum_{t \in T} \sum_{r \in \mathcal{R}} (U^c(r,t) + U^m(r,t))}{2 \times |T| \times |\mathcal{R}|} \tag{12}$$

$$U^c(r,t) = \frac{\sum_{\forall y(\alpha)=r} \nu^c(\alpha,t)}{\Gamma^c(r)}, \tag{13}$$

$$U^m(r,t) = \frac{\sum_{\forall y(\alpha)=r} \nu^m(\alpha,t)}{\Gamma^m(r)}, \tag{14}$$

Equations (13) and (14) are the CPU-based resource utilization and memory-based resource utilization, respectively. The total resource utilization is also an important metric to measure the performance of the algorithm.

Table 4 shows the mean and standard deviation of the total energy consumption on the test data of 30 independent runs of CCGP and baseline methods for different scenarios. Overall, CCGP is the best algorithm because it significantly outperforms other algorithms on all scenarios and always gets the least energy consumption. Random and Binpacking are not significant on several scenarios, furthermore, CCGP is over 30% more energy efficient than either of these two default scheduling strategies. For the training scenario based on <20, 0.2>, we can observe that the scheduling heuristic obtained

by CCGP evolution still has strong adaptive ability with the change of the number of workflows. For the training scenario based on <20, 0.5>, we can observe that the scheduling heuristic obtained by CCGP evolution still has strong adaptive ability with the change of arrival rate. Compared to NCGP, CCGP saves 20% energy consumption, suggesting that cooperative coevolution among populations can improve population optimization search.

Table 4. The mean (standard deviation) total energy consumption on the test data of 30 independent runs of CCGP and baseline methods for different scenarios

Training Scenario	<wfNum, λ>*	Random	Binpacking	NCGP	CCGP
<20, 0.2>	<10, 0.2>	6.54(3.60)	6.00(3.17)(−)	2.41(0.44)(−)(−)	1.80(0.08)(−)(−)(−)
	<20, 0.2>	13.70(6.05)	13.44(6.96)(≈)	5.50(0.95)(−)(−)	4.26(0.71)(−)(−)(−)
	<30, 0.2>	19.74(8.38)	19.33(7.93)(≈)	7.48(1.32)(−)(−)	5.63(0.31)(−)(−)(−)
	<40, 0.2>	28.25(8.10)	28.18(8.11)(≈)	12.08(1.83)(−)(−)	9.45(0.44)(−)(−)(−)
<20, 0.5>	<20, 0.2>	13.70(6.05)	13.44(6.96)(≈)	5.44(0.91)(−)(−)	4.25(0.72)(−)(−)(−)
	<20, 0.5>	13.60(6.12)	13.43(6.04)(≈)	5.39(0.92)(−)(−)	4.08(0.17)(−)(−)(−)
	<20, 0.8>	14.26(6.60)	13.06(6.61)(−)	5.41(0.89)(−)(−)	4.12(0.18)(−)(−)(−)
	<20, 1.0>	13.92(6.57)	13.02(6.11)(≈)	5.40(0.89)(−)(−)	4.12(0.19)(−)(−)(−)

* wfNum is the total number of workflows submitted and λ is the arrival rate

Table 5 shows the mean and standard deviation of the total resource utilization on the test data of 30 independent runs of CCGP and baseline methods for different scenarios. Overall, CCGP is the best algorithm because it significantly outperforms other algorithms on all scenarios and always achieves the highest resource utilization. The results also show that for all scenarios, the resource utilization of all algorithms does not vary with the change of workflow or the change of arrival rate. CCGP also improves resource utilization by 20% compared to NCGP, which is consistent with the saved energy consumption. Although Binpacking greedily selects resources with high resource utilization to deploy tasks, one possible reason for its lower resource utilization is that it does not have a better balance of both CPU and memory dimensions at the same time, which can result in more resources being requested.

Table 5. The mean (standard deviation) total resource utilization on the test data of 30 independent runs of CCGP and baseline methods for different scenarios

Training Scenario	<wfNum, λ>	Random	Binpacking	NCGP	CCGP
<20, 0.2>	<10, 0.2>	0.17(0.04)	0.15(0.04)(+)	0.47(0.12)(−)(−)	0.59(0.03)(−)(−)(−)
	<20, 0.2>	0.17(0.03)	0.15(0.02)(+)	0.47(0.11)(−)(−)	0.60(0.03)(−)(−)(−)
	<30, 0.2>	0.15(0.03)	0.14(0.03)(+)	0.49(0.13)(−)(−)	0.61(0.04)(−)(−)(−)
	<40, 0.2>	0.16(0.02)	0.14(0.02)(+)	0.50(0.13)(−)(−)	0.62(0.04)(−)(−)(−)
<20, 0.5>	<20, 0.2>	0.17(0.03)	0.15(0.02)(+)	0.48(0.11)(−)(−)	0.61(0.03)(−)(−)(−)
	<20, 0.5>	0.16(0.03)	0.15(0.03)(≈)	0.48(0.11)(−)(−)	0.61(0.03)(−)(−)(−)
	<20, 0.8>	0.16(0.02)	0.15(0.03)(≈)	0.48(0.11)(−)(−)	0.61(0.03)(−)(−)(−)
	<20, 1.0>	0.16(0.02)	0.15(0.02)(+)	0.48(0.10)(−)(−)	0.61(0.03)(−)(−)(−)

6 Conclusions

This paper establishes an energy-efficient dynamic workflow scheduling model that considers the impact of finer-grained tasks with both CPU and memory configurations on energy consumption. We develop a dynamic workflow scheduling simulator to mimic the scheduling process in real-world scenarios, which employs three scheduling heuristics to make decisions at three decision points to keep scheduling going. The proposed CCGP can evolve both the task selection rule and the instance selection rule simultaneously to meet these decision points. In the experimental study, the real-world cloud workflows traces are used as the training and test cases. The simulation results show that the high-level heuristics evolved by the proposed CCGP algorithm can improve more than 20% in terms of energy consumption and resource utilization. In the future, we plan to extend our method by utilizing multi-objective optimization techniques to optimize both cost and energy consumption of the heuristics. Furthermore, we will investigate the impact of different types of workflows on the performance of the scheduling heuristics learned by the CCGP, e.g., in terms of complexity, data transfer requirements, or task execution time.

Acknowledgments. This work was supported by the Shenzhen Science and Technology Program under Grant No.GXWD20220817124827001, and No. JCYJ20210324132406016.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Katal, A., Dahiya, S., Choudhury, T.: Energy efficiency in cloud computing data centers: a survey on software technologies. Clust. Comput. **26**, 1845–1875 (2023)

2. Rodriguez, M.A., Buyya, R.: Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. Futur. Gener. Comput. Syst. **79**, 739–750 (2018)
3. Ye, L., Xia, Y., Yang, L., Yan, C.: SHWS: stochastic hybrid workflows dynamic scheduling in cloud container services. IEEE Trans. Autom. Sci. Eng. **19**, 2620–2636 (2022)
4. Havet, A., Schiavoni, V., Felber, P., Colmant, M., Rouvoy, R., Fetzer, C.: GENPACK: a generational scheduler for cloud data centers. In: 2017 IEEE International Conference on Cloud Engineering, pp. 95–104. IEEE (2017)
5. Hu, B., Cao, Z., Zhou, M.: Scheduling real-time parallel applications in cloud to minimize energy consumption. IEEE Trans. Cloud Comput. **10**, 662–674 (2022)
6. Stavrinides, G.L., Karatza, H.D.: An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations. Futur. Gener. Comput. Syst. **96**, 216–226 (2019)
7. Fan, G., Chen, X., Li, Z., Yu, H., Zhang, Y.: An energy-efficient dynamic scheduling method of deadline-constrained workflows in a cloud environment. IEEE Trans. Netw. Serv. Manage. **20**, 3089–3103 (2023)
8. Sun, Z., Li, Z., Gu, C., Huang, H.: An energy-efficient scheduling method for real-time multi-workflow in container cloud. In: Wu, W., Guo, J. (eds.) COCOA 2023. LNCS, vol. 14461, pp. 168–181. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-49611-0_12
9. Burke, E.K., et al.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Socy. **64**, 1695–1724 (2013)
10. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. IEEE Trans. Evol. Comput. **28**, 147–167 (2023)
11. Miyashita, K.: Job-shop scheduling with genetic programming. In: Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation. pp. 505–512. Morgan Kaufmann Publishers Inc., San Francisco (2000)
12. Zhang, F., Nguyen, S., Mei, Y., Zhang, M.: Genetic Programming for Production Scheduling. Springer, Singapore (2021). https://doi.org/10.1007/978-981-16-4859-5
13. Xu, M., et al.: Genetic programming for dynamic workflow scheduling in fog computing. IEEE Trans. Serv. Comput. **16**, 2657–2671 (2023)
14. Yang, Y., Chen, G., Ma, H., Zhang, M.: Dual-tree genetic programming for deadline-constrained dynamic workflow scheduling in cloud. In: Troya, J. et al. (eds.) ICSOC 2022. LNCS, vol. 13740, pp. 433–448.Springer, Cham (2022). https://doi.org/10.1007/978-3-031-20984-0_31
15. Sun, Z., Zhang, B., Gu, C., Xie, R., Qian, B., Huang, H.: ET2FA: a hybrid heuristic algorithm for deadline-constrained workflow scheduling in cloud. IEEE Trans. Serv. Comput. **16**, 1807–1821 (2023)
16. Zhu, Q.H., Tang, H., Huang, J.J., Hou, Y.: Task scheduling for multi-cloud computing subject to security and reliability constraints. IEEE/CAA J. Automatica Sinica. **8**, 848–865 (2021)
17. Xiao, Q.Z., Zhong, J., Feng, L., Luo, L., Lv, J.: A cooperative coevolution hyper-heuristic framework for workflow scheduling problem. IEEE Trans. Serv. Comput. **15**, 150–163 (2022)
18. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via co-operative coevolution genetic programming. IEEE Trans. Evol. Comput. **18**, 193–208 (2014)
19. Tuli, S., Casale, G., Jennings, N.R.: MCDS: AI augmented workflow scheduling in mobile edge cloud computing systems. IEEE Trans. Parallel Distrib. Syst. **33**, 2794–2807 (2022)
20. Ma, X., Zhou, A., Zhang, S., Li, Q., Liu, A.X., Wang, S.: Dynamic task scheduling in cloud-assisted mobile edge computing. IEEE Trans. Mob. Comput. **22**, 2116–2130 (2023)

21. Sun, Z., Huang, H., Li, Z., Gu, C., Xie, R., Qian, B.: Efficient, economical and energy-saving multi-workflow scheduling in hybrid cloud. Expert Syst. Appl. **228**, 120401 (2023)
22. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**, 260–274 (2002)