IEEE INFOCOM 2026 - IEEE Conference on Computer Communications | 979-8-3315-4961-9/26/$31.00 ©2026 IEEE | DOI: 10.1109/INFOCOM59046.2026.11571438

Multitask Cooperative Genetic Programming for Co-Scheduling Online-Offline Workflows in the Cloud

Zaixing Sun*, Liang Zhang†, Quan Tang*, Jun Jiang*, Chonglin Gu‡, and Bin Wang*
*Pengcheng Laboratory, Shenzhen, China
†University of Science and Technology Beijing, Beijing, China, and Key Laboratory of New Power System Operation Control of Guizhou Province, School of Data Science, Guizhou Institute of Technology, Guiyang, China
‡Harbin Institute of Technology (Shenzhen), Shenzhen, China
zaixing.sun@micc.hitsz.edu.cn, rushup@foxmail.com, tangq@pcl.ac.cn, jiangj01@pcl.ac.cn, guchonglin@hit.edu.cn, wangb02@pcl.ac.cn

***Abstract*—The underutilization of cloud resources remains a significant challenge due to overprovisioning and resource silos. While co-scheduling latency-critical online workflows with best-effort offline workflows is a promising strategy, existing approaches often overlook the complex bidirectional coupling and fine-grained resource dependencies between these workloads. This paper studies the Co-Scheduling of Online-Offline Workflows problem, formulated as a coupled multitask optimization model that jointly accounts for fine-grained CPU/memory configuration and parallel task execution within virtual machines. The objective is to simultaneously minimize the total flowtime of online workflows and the total cost of shared cloud clusters. We propose a Multitask Cooperative Genetic Programming (MCGP) approach to address the dynamic interaction where the scheduling decision of one task alters the environment of another. MCGP automatically evolves and learns two dedicated rule pairs, comprising task and resource selection rules, tailored for online and offline workflows, respectively. MCGP enables the two scheduling strategies to collaborate effectively within a shared environment by integrating multitask learning for knowledge transfer and cooperative coevolution for strategy co-adaptation. Extensive simulation experiments based on real-world traces show that MCGP consistently outperforms existing baselines in terms of reducing total flowtime and costs, and improving the success rate.**

***Index Terms*—Cloud computing, workflow scheduling, co-scheduling, genetic programming, multitask learning**

I. Introduction

With cloud computing becoming critical digital infrastructure, workload optimization and waste reduction have emerged as top priorities for large enterprises with large cloud spend in recent years [1]. Gartner forecasts global public cloud end-user spending will total $723 billion in 2025 [2], yet organizations estimate nearly 27% of their cloud costs on IaaS and PaaS are wasted [3]. There are two main factors that cause these wastes. Firstly, the over-provisioning strategy, which is widely adopted to ensure quality of service (QoS) during peak periods for online services, directly results in a large number of idle resources [4], [5]. Secondly, resource siloing hinders the effective reuse of these idle resources, particularly in traditional siloed deployments where different workload types are deployed on separate clusters [6]. Together, these factors lead to the long-term underutilization of data center resources.

To address the aforementioned resource efficiency challenges, online-offline workload hybrid deployment (co-location) is a key industry strategy to improve the utilization rate of shared cluster resources [7]. The effectiveness of this strategy stems from the strong complementarity between two types of workloads. Online workloads are latency-critical services with dynamic and sudden resource demands. They are constrained by strict service-level agreements and focus on response time and QoS. In contrast, offline workloads are best-effort batch jobs that are latency-insensitive and can tolerate preemption. They focus on throughput or cost-effectiveness. The core of hybrid deployment is to dynamically allocate underutilized cluster resources to offline workloads, thereby improving overall resource utilization and saving costs. Ye et al. [8] proposed the SHWS to address stochastic hybrid workflow scheduling, utilizing sub-deadline assignment and task prioritization to minimize costs while meeting deadline constraints. To safeguard service quality, Zhu et al. developed Toposch [9] and Prank [10], which employ critical path analysis and PageRank-weighted anomaly localization to protect the QoS of distributed long-running applications while harvesting idle resources for batch jobs. Despite these advancements, existing methods often overlook the intricate bi-directional coupling and the fine-grained resource dependencies between online and offline workloads.

This paper studies the Co-Scheduling of Online-Offline Workflows (CSO2W), a coupled multitask optimization problem that aims to minimize the total flowtime of deadline-constrained online workflows and the total cost of the shared cluster. Its complexity arises not only from inheriting classic workflow scheduling challenges [11], but more critically from the bidirectional dynamic coupling between the two

This work was supported in part by the Major Key Project of PCL, China under Grant PCL2025A13, Qiankehe Platform under Grant ZSYS[2025]007, Shenzhen Science and Technology Program under Grant GXWD20220817124827001, and No.SYSPG20241211173609009. (Corresponding author: Bin Wang)

$task$[1] types: scheduling decisions for offline $task$s introduce dynamic resource contention that reshapes the environment for online $task$s in real time, and vice versa. To address this 'solution-as-environment' interdependence and the associated 2-dimensional bin-packing problem in resource selection, we propose a Multitask Cooperative Genetic Programming (MCGP) approach to automatically co-evolve four dedicated decision rules: subtask prioritization and resource selection rules for both online and offline workflows. MCGP integrates multitask learning for knowledge transfer [12], cooperative co-evolution for strategy co-adaptation [13], and genetic programming hyper-heuristic [14] for automated rule discovery. This framework enables $task$-specific optimization where latency-sensitive online scheduling and efficiency-oriented offline scheduling collaborate effectively within the same dynamic environment. The main contributions are as follows:

- We provide a coupled $multitask$ optimization model for the co-scheduling online-offline workflow in cloud, which jointly considers fine-grained CPU and memory configuration of subtasks and a VM supporting the parallel execution of multiple tasks simultaneously.
- We propose an MCGP method that automatically evolves and learns two dedicated rule pairs for online and offline workflows, where each $task$ category is governed by its own subtask selection and resource selection rules.
- Extensive simulation experiments using the real-world data traces demonstrate that the proposed algorithm significantly outperforms existing baselines.

II. Co-Scheduling Workflows Model

We formulate the co-scheduling online-offline workflow in cloud as a coupled $multitask$ optimization problem. We will first introduce the common constraints shared by both online and offline $task$s, followed by the specific models for each $task$ category.

Workflow applications are categorized as either online or offline and submitted to the cloud dynamically over time. Let $\mathcal{G}$ be a set of workflow applications. Let $\rho^c(g) \in \{online, offline\}$, $\rho^a(g)$, and $\rho^d(g)$ be the category, arrival time, and deadline of workflow $g \in \mathcal{G}$, respectively. Each workflow g contains a set of subtasks $A(g)$. Each subtask $\alpha \in A(g)$ is represented by a tuple $\{\tau^e(\alpha), \mu^c(\alpha), \mu^m(\alpha), A^{pre}(\alpha), A^{suc}(\alpha)\}$, where $\tau^e(\alpha)$ is the reference execution time of the subtask, $\mu^c(\alpha)$ is minimum computational resource requirement, $\mu^m(\alpha)$ is minimum memory resource requirement, $A^{pre}(\alpha) \subseteq A(g)$ is a set of immediate predecessors (parent subtasks) of subtask α, and $A^{suc}(\alpha) \subseteq A(g)$ is a set of immediate successors (child subtasks) of subtask α. $d(\alpha_1, \alpha_2)$ is the data required to be transferred from subtask α_1 to its successor subtask $\alpha_2 \in A^{suc}(\alpha_1)$. We consider a computing resource in the cloud environment to be a host. Let $\mathcal{R}$ be the set of heterogeneous computing resources. Each resource r is associated with a type $\theta(r) \in \Theta$, which contains the parameters the size of CPU $\Gamma^c(r)$, the size of memory $\Gamma^m(r)$, the per-unit lease price $c^e(r)$, and the price of transmitting unit data $c^d(r)$. Let $\vec{b}$ be the network bandwidth of resources. The subtasks of workflow are packaged as containers through existing image files, and then the containers are deployed to the host. A host can deploy multiple containers simultaneously, but cannot exceed the configurations of the host. Containers are minimal units, i.e. a container can only run one subtask at a time.

We assume scheduling as a discrete-time control problem as standard in previous work [15], [16]. Let T be the set of decision points, and $t \in T$ be a specific decision point. Decision points typically correspond to moments when the system's state changes, such as when an online/offline workflow is submitted, or a subtask starts execution. Let $x^s(\alpha)$ and $x^f(\alpha)$ be the execution start time and execution finish time of the subtask α. Let $y(\alpha) \in \mathcal{R}$ be assigned resource of the subtask α in the schedule. Let $\nu^c(\alpha, t)$ and $\nu^m(\alpha, t)$ be the sizes of the CPU and memory allocated to the container used to execute the subtask α at time t.

Common constraints: Since different categories of $task$s are processed within a single system, they are subject to the following common constraints.

- The system does not have any information about the workflow until it is submitted, as shown in Eq. (1).
- A subtask cannot start executing until all of its predecessor subtasks have been completed, as shown in Eq. (2). Eq. (3) is the communication time between each subtask α_1 and its successor subtask α_2.
- Each subtask is executed by a container that matches its CPU and memory configuration, as shown in Eq. (4).
- Each resource can deploy and run multiple containers simultaneously, but the sum of the CPU and memory capacities of the containers allocated on the resource cannot exceed the corresponding resource's capacity at any moment, as shown in Eq. (5).

$$x^s(\alpha) \geq \rho^a(g),\ \forall\ g \in \mathcal{G},\ \alpha \in A(g), \tag{1}$$

$$x^s(\alpha_1) + \tau^e(\alpha) + x^c(\alpha_1, \alpha_2) \leq x^s(\alpha_2), \forall \alpha_2 \in A^{suc}(\alpha_1), \tag{2}$$

$$x^c(\alpha_1, \alpha_2) = \begin{cases} 0, & y(\alpha_1) = y(\alpha_2), \\ d(\alpha_1, \alpha_2)/\vec{b}, & \text{otherwise}, \end{cases} \tag{3}$$

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

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

Online $Task$ Model: Online workflows are latency-critical applications that require strict QoS guarantees. We use the total flowtime of online workflows to measure the QoS level. The response time of an online workflow g is defined as the duration from its arrival time to the completion time of all its subtasks. The total flowtime of all online workflows is

[1]To avoid ambiguity in this paper, we use the term $task$ to denote a task in the context of multitask optimization, and subtask to refer to a task within a workflow.

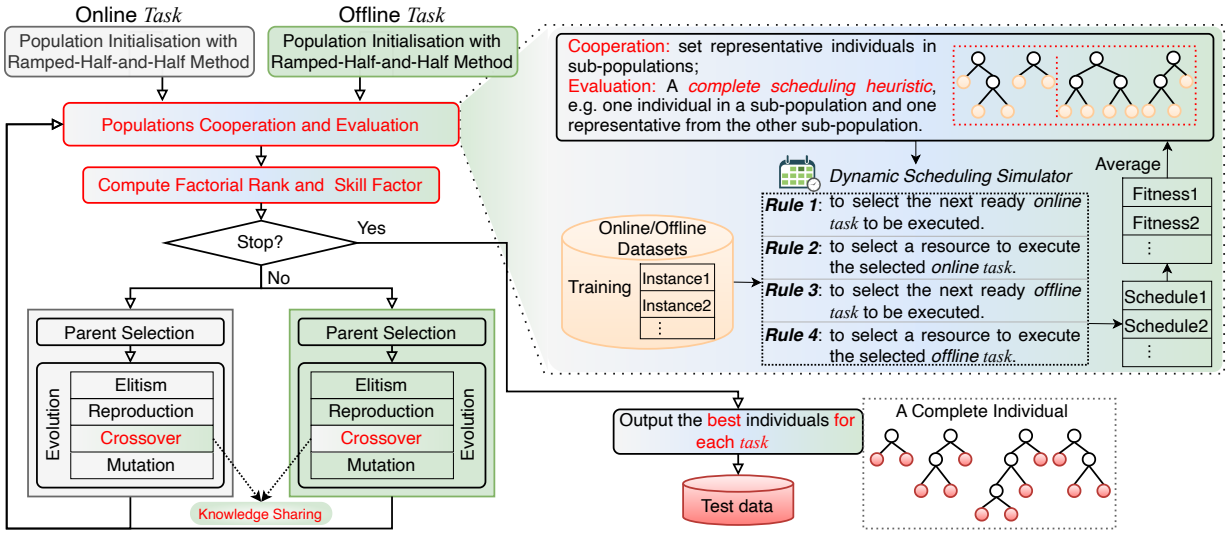

Fig. 1. The flowchart of the multitask cooperative genetic programming approach.

formulated in Eq. (6).

$$\min \mathbb{T} = \sum_{\substack{\forall g \in \mathcal{G}, \\ \rho^c(g)=online}} \left(\max_{\alpha \in A(g)} x^f(\alpha) - \rho^a(g) \right), \quad (6)$$

Offline Task Model: Offline workflows are best-effort applications that prioritize system-wide cost-effectiveness. Their performance is measured by the total monetary cost, which includes the lease cost of resources and the data communication cost between subtasks, as formulated in Eq. (7).

$$\min \ \mathbb{C} = \sum_{r \in \mathcal{R}} \int_T C^e(r,t)dt + \sum_{g \in \mathcal{G}} \sum_{\substack{\alpha_1 \in A(g), \\ \alpha_2 \in A^{suc}(\alpha_1)}} C^d(\alpha_1, \alpha_2), \quad (7)$$

$$s.t.: \ U^c(r,t) = \frac{\sum_{\forall y(\alpha)=r} \nu^c(\alpha,t)}{\Gamma^c(r)}, \quad (8)$$

$$U^m(r,t) = \frac{\sum_{\forall y(\alpha)=r} \nu^m(\alpha,t)}{\Gamma^m(r)}, \quad (9)$$

$$C^e(r,t) = \begin{cases} c^e(r), & U^c(r,t) \times U^m(r,t) \neq 0, \\ 0, & \text{otherwise}, \end{cases} \quad (10)$$

$$C^d(\alpha_1,\alpha_2) = \begin{cases} c^d(r) \times d(\alpha_1,\alpha_2), & \text{if } y(\alpha_1) \neq y(\alpha_2), \\ 0, & \text{otherwise}. \end{cases} \quad (11)$$

Eqs. (8) and (9) are the CPU-based resource utilization and memory-based resource utilization, respectively. Eq. (10) indicates the resources in the cloud server will be charged when running the subtasks. Eq. (11) indicates that data will be charged when it is transmitted from the cloud server.

III. Multitask Cooperative Genetic Programming

Fig. 1 shows the flowchart of using MCGP approach. The input is two categories of online and offline $tasks$ to be solved. The output is two learned best complete scheduling heuristic rules, each of which is specifically focused on the corresponding $task$. We use two sub-populations to solve these two $tasks$ simultaneously through collaboration and knowledge sharing. The MCGP approach automatically generates and learns the following four heuristic rules to make the corresponding decisions during the scheduling process.

Online subtask selection rule (Rule 1): to select the next ready online subtask to be executed. The rule is used to determine the sequence of subtasks if multiple subtasks are ready when an online subtask is completed or a new online workflow is submitted.

Resource selection rule for online subtask (Rule 2): to select an appropriate resource in cloud to execute the selected online subtask. There are three main steps. First of all, eliminating the resources that con not meet the minimum requirements of CPU or Memory of the subtask. Secondly, it filters out the resources that cannot meet the sub-deadline to execute the subtask. If no resource remains, then the resource in the cloud with the earliest finish time is selected to execute the subtask from all available resources. Otherwise, the rule is used to select a resource. In cases where multiple resources have the same priority, we prioritise selecting an existing resource that does not need to be created. In this rule, the available resources include leased and non-terminated VMs set, as well as non-leased VMs set with all VM types.

Offline subtask selection rule (Rule 3): to generate priorities for ready offline subtasks and add them to the offline subtask queue according to their priorities. The rule is used when an offline subtask is completed or a new offline workflow is submitted.

Resource selection rule for offline subtask (Rule 4): to select an appropriate resource in cloud to execute the selected offline subtask. We also need to first filter out the resources that cannot meet the minimum requirements of CPU or Memory of the subtask. Then, the rule is used to select a resource from the available resources to execute the subtask. In this rule, the available resources include only leased and non-terminated VMs set.

A. Evolutionary Mechanism

Populations Initialization with Ramped-Half-and-Half Method. Initially, individuals in the two subpopulations are generated using the ramped-half-and-half method, where terminals and functions are randomly selected and combined. Specifically, half of the individuals in a subpopulation are initialized with the maximum depth set in advance, while the remaining individuals in the subpopulation are initialized randomly within the maximum depth.

Populations Cooperation and Evaluation. Given a training instance, we can evaluate the fitness of an individual by applying it to the simulator, which is used to imitate the scheduling process, such as submitting workflows, executing subtasks, and transferring data. After evaluation, each individual will get the two objective values considered. In each generation, a representative individual is first selected from each subpopulation. Each individual within a subpopulation then cooperates with these representatives from the another subpopulation to form a complete scheduling heuristic for making four decisions. This complete scheduling heuristic is applied to the simulator to evaluate the fitness of the individual. Specifically, representative individuals are selected randomly from each subpopulation in the first generation. In subsequent generations, we randomly select a representative from the elitism individuals of each subpopulation.

Compute Factorial Rank and Skill Factor. We employ the concept of factorial rank and skill factor from multifactorial evolutionary algorithm [12] to guide the evolution of individuals in different subpopulations. Based on individual fitness, we first get the ranking in different $task$s. The ranking value of the $task$ with higher ranking is taken as the individual's skill fitness.

Parent Selection. If the stopping criterion is met, the best individuals for different $task$s so far are considered as the best evolved scheduling heuristics for CSO2W. Otherwise, we use the tournament selection to select parents based on the skill fitness of the individuals for generating offspring.

Evolution. Evolution has four genetic operators, which are elitism, reproduction, crossover and mutation. These operators aim at generating a new offspring population by inheriting good materials from the parent population. For elitism, the top predefined number of elite individuals are directly copied into the next generation. Mutation involves replacing a randomly selected subtree of the parent with a newly generated subtree created using the same initialization rules. Crossover involves randomly selecting a subtree from a parent and replacing it with a subtree extracted from an elite individual, who is randomly sampled from the elite sets of both subpopulations. This process promotes inter-$task$ knowledge sharing by leveraging genetic information from elites of all $task$s.

B. Representation, Terminal Set, and Function Set

To evolve a scheduling heuristic with four rules for the CSO2W problem, we adopt a tree-based representation, where each individual consists of two trees. Each tree represents a scheduling rule for a specific decision-making process. we design 25 terminals based on the scheduling process, which indicate the characteristics related to workflows, subtasks, and resources, to describe the state of the cloud. Table I shows the terminal and function sets of MCGP. We use the symbol '✓' to mark the terminals that compose the corresponding rules in Table I. The function set is {+, -, *, protected /, Max, Min}, and each function has two arguments. The protected division "/" returns one if divided by zero. Below we provide additional explanations for those terminals in the table that are not self-explained.

TABLE I
TERMINAL AND FUNCTION SETS OF MCGP.

No.	Nation	Description	Rules 1&3	Rules 2&4
1	$CPUA$	The minimum CPU requirement of a subtask.	✓	
2	$MEMA$	The minimum memory requirement of a subtask.	✓	
3	ET	The execution time of a subtask on a reference resource.	✓	
4	SD	The sub-deadline of a subtask.	✓	
5	UR	The upward rank of a subtask.	✓	
6	VCT	The average communication time for a subtask.	✓	
7	NKQ	The number of subtasks in ready queue.	✓	
8	NRA	The number of remaining (or unscheduled) subtasks in a workflow.	✓	
9	$TETQ$	The total ET of subtasks in ready queue.	✓	
10	$TETRA$	The total ET of remaining subtasks in a workflow.	✓	
11	NPA	The number of direct predecessors for a subtask.	✓	
12	NSA	The number of direct successors for a subtask.	✓	
13	$CPUR$	The CPU size of a resource.		✓
14	$MEMR$	The memory size of a resource.		✓
15	PR	The unit price of a resource.		✓
16	CTR	The communication time of a subtask on a resource.		✓
17	$ETAR$	The execution time of a subtask on a resource.		✓
18	AAT	The actual available time of a resource.		✓
19	ST	The slack time of subtask.		✓
20	$ECAR$	The execution cost of a subtask on a resource.		✓
21	IC	The idle CPU of a resource.		✓
22	IM	The idle memory of a resource.		✓
23	UC	The CPU utilization of a resource.		✓
24	UM	The memory utilization of a resource.		✓
25	CC	The communication cost of a subtask while it is executed on a resource.		✓
Function set		+, -, *, protected /, Max, Min	✓	✓

No.4 SD is the sub-deadline of a subtask, which is set based on the latest finish time, as shown in Eq. (12). We use reference execution time of a subtask α (e.g., terminal ET, denote as $ET(\alpha)$) and the communication time to estimate the latest finish time $\ell(\alpha)$ (Eq. (13)). ε is a random factor within the range $[0.95, 1]$ [17]. No.5 UR is the upward rank of a subtask α, which is the length of the critical path from α to the exit subtask [18], as shown in Eq. (14).

$$SD(\alpha) = \begin{cases} \ell(\alpha), \text{ if } A^{suc}(\alpha) = \varnothing, \\ \varepsilon \times \ell(\alpha), \quad \text{otherwise.} \end{cases} \tag{12}$$

$$\ell(\alpha_1) = \begin{cases} \rho(g), & \text{if } A^{suc}(\alpha_1) = \varnothing, \\ \max\limits_{\alpha_2 \in A^{suc}(\alpha_1)} \left\{ \ell(\alpha_2) - ET(\alpha_2) - \frac{d(\alpha_1,\alpha_2)}{\bar{b}} \right\}, & \text{otherwise.} \end{cases} \tag{13}$$

$$UR(\alpha_1) = \begin{cases} ET(\alpha_1), & \text{if } A^{suc}(\alpha_1) = \varnothing, \\ ET(\alpha_1) + \max\limits_{\alpha_2 \in A^{suc}(\alpha_1)} \left\{ UR(\alpha_2) + \frac{d(\alpha_1,\alpha_2)}{\bar{b}} \right\}, & \text{otherwise.} \end{cases} \tag{14}$$

No.16 CTR is the communication time of a subtask while it is executed on one resource, which is the time at which all predecessors' data is finally received. No.18 AAT is the

actual available time of a resource for a subtask and is equal to the maximum between the current system time plus the latency of the resource and CTR. When this resource is newly created, its latency is replaced by the booting time. For online workflow, the slack time No.19 ST is $(AAT - SD)$. For offline workflow, ST is the non-negative difference between its earliest available completion time and the latest completion time among all subtasks assigned to the instance. In other words, the ST of an offline subtask is the extra time needed to execute that subtask via the resource. Multiplying this value by the rental unit price PR yields the execution cost No.20 $ECAR$. No.21 IC and No.22 IM are calculated based on the sum of the requirements of running subtasks and waiting subtasks on the resource.

IV. Experiments and Discussions

A. Environmental Setting

In simulation experiment, we use 5 representative VM types from low configuration to high configuration. The VM configurations are based on current Amazon EC2 platform, as presented in Table II. The bandwidths is set to 2.0 GB/s. The unit price of data transmission is set to 0.2 $/GB. The booting time of VMs is set to 55.9s. The system maintains an elastic VM pool with no upper limit. Instances idle for over 0.5 hours are automatically terminated, while a minimum of five VMs is retained to ensure rapid response to incoming $task$s.

We consider the cluster data released by Alibaba Cloud in 2018. The input and output data of each subtask is assigned by a uniform discrete distribution between 500 and 5000 MB. Each online workflow's deadline factor is randomly selected from the set $\{1.2, 1.5, 1.8, 2.0\}$. Workflow arrival is modelled using the Poisson distribution with an arrival rate λ where the interarrival time is exponentially distributed with $1/\lambda$, where $\lambda \in \{0.005, 0.01, 0.015\}$ [8]. The name of each scenario is set to $\langle \mathrm{wfNum}, \lambda \rangle$, where wfNum is the number of workflows submitted for each category (online and offline).

We compare the proposed approach with three benchmarks to verify the efficacy. Binpacking is one of the default scheduling strategies supported in Docker Swarm's initial release, which executes a container on the most compacted node regarding resources in order to reduce the total number of used nodes. SHWS is a scheduling heuristic for offline batch workflows and online stream workflows in cloud [8], but it only considers that a VM only performs one subtask at a time. To assess the impact of co-evolution, we compare MCGP with its variant, MGP, where each individual directly encodes all four rules in a single population. Since GP yield distinct optimal individuals for different $task$s, we denote MCGP's offline- and online-oriented best individuals as MCGP$^-$ and MCGP$^+$, respectively. The notation for MGP follows the same convention (i.e., MGP$^-$ and MGP$^+$). The parameters of CCGP are shown in Table III. The population size of MGP is set to 400 to keep the same total population size as MCGP.

We implement the simulation and all the compared algorithms in Python 3.10 and C++ and deploy them on a computer with 2.20 GHz Intel Xeon CPU E5-2630 and 128 GB of memory. For all the compared algorithms, 30 independent runs are done for each scenario and the evolved scheduling heuristics are tested on 50 unseen instances. The average objective value across the 50 test instances is reported as the test performance of the rule.

TABLE II
Virtual Machine's Types and Configurations

Type	vCPU	ECU	Memory (GB)	Cost($/hour)
c3.large	2	7	3.75	0.132
c3.xlarge	4	14	7.5	0.265
c3.2xlarge	8	28	15	0.529
c3.4xlarge	16	55	30	1.058
c3.8xlarge	32	108	60	2.117

TABLE III
The parameter settings of MCGP.

Parameter	Value
Population size	2 subpopulations and 200 for each subpopulation
Number of generations	51
Method for initialising population	Ramped-half-and-half
Initial minimum/maximum depth	2 / 7
Elitism	10
Maximal depth	8
Crossover/mutation/reproduction rate	0.8 / 0.15 / 0.05
Parent selection	Tournament selection with size 7
Terminal/non-terminal selection rate	10% / 90%

B. Performance Results

Although MCGP increases offline training time to 1.9 times that of MGP due to its co-evolutionary mechanism, it maintains microsecond-level online responsiveness by leveraging symbolic heuristics that match the execution latency of MGP.

Tables IV and V show the mean (standard deviation) of the total flowtime, total cost, and success rate over the 30 independent runs of the compared algorithms, respectively. Overall, the proposed MCGP framework demonstrates significant competitive advantages across all core performance metrics. Friedman and Nemenyi post-hoc tests over 30 independent runs indicate that both MCGP$^+$ and MCGP$^-$ consistently maintain leading positions in their respective $task$ orientations. Table IV shows that MCGP$^+$ and MCGP$^-$ achieve the lowest average ranks of 1.417 for Total Flowtime and 1.144 for Total Cost, respectively. In contrast, traditional algorithms, such as Binpacking and SHWS, exhibit poor performance (often ranking lowest) as they fail to account for the intricate bidirectional coupling between online and offline workflows. The superiority of MCGP is further evidenced by the success rate results in Table V, where MCGP$^+$ achieves near-perfect performance (99.9%–100%) across all scenarios. MCGP significantly surpasses the SHWS algorithm (average rank 6.000), which was originally designed for a single-task execution paradigm and thus lacks the scheduling mechanisms to effectively manage the resource sharing inherent in the parallel execution environment considered here.

Furthermore, comparing MCGP and MGP highlights the impact of cooperative coevolution. MCGP decomposes decision rules into co-evolving subpopulations, enabling the simultane-

TABLE IV
THE MEAN AND STANDARD DEVIATION OF THE OBJECTIVES OVER THE 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS.

Scenarios	Total Flowtime						Total Cost					
$\langle \text{wfNum}, \lambda \rangle$	Binpacking	SHWS	MGP$^-$	MGP$^+$	MCGP$^-$	MCGP$^+$	Binpacking	SHWS	MGP$^+$	MGP$^-$	MCGP$^+$	MCGP$^-$
$\langle 100, 0.005 \rangle$	85.96/0.03	88.58/0.02	87.05/2.34	85.32/0.06	86.49/1.25	85.31/0.01(-)(-)(-)(=)(-)	464.45/35.78	650.16/1.50	478.38/242.26	241.85/67.16	383.79/213.59	161.25/60.99(-)(-)(-)(-)(-)
$\langle 200, 0.005 \rangle$	152.65/0.04	157.59/0.03	148.84/4.72	145.39/0.07	147.66/2.45	145.37/0.01(-)(-)(-)(=)(-)	937.39/45.57	1305.03/2.36	924.60/462.60	525.12/173.91	723.27/409.40	348.09/128.09(-)(-)(-)(-)(-)
$\langle 100, 0.01 \rangle$	86.04/0.03	88.34/0.03	86.47/1.81	85.33/0.06	86.64/1.92	85.31/0.01(-)(-)(-)(-)(-)	587.92/23.63	674.37/1.41	610.59/244.74	272.40/58.10	462.11/225.95	188.87/27.93(-)(-)(-)(-)(-)
$\langle 200, 0.01 \rangle$	152.74/0.04	156.98/0.03	147.65/3.67	145.40/0.07	147.89/3.79	145.38/0.01(-)(-)(-)(-)(-)	1203.87/33.78	1349.53/2.05	1193.69/457.24	551.20/125.74	877.24/441.64	382.81/55.98(-)(-)(-)(-)(-)
$\langle 100, 0.015 \rangle$	86.08/0.03	88.27/0.03	86.12/1.73	85.33/0.02	86.24/1.47	85.34/0.10(-)(-)(-)(+)(-)	639.76/19.08	685.15/1.44	574.33/215.58	295.71/39.02	586.80/254.40	218.95/27.77(-)(-)(-)(-)(-)
$\langle 200, 0.015 \rangle$	152.80/0.04	156.74/0.03	146.96/3.53	145.40/0.02	147.33/3.08	145.63/1.27(-)(-)(-)(+)(-)	1313.71/25.89	1366.51/1.61	1123.56/416.78	591.68/99.93	1124.90/491.96	451.19/53.49(-)(-)(-)(-)(-)
W/D/L	0/0/6	0/0/6	0/0/6	2/2/2	0/0/6	N/A	0/0/6	0/0/6	0/0/6	0/0/6	0/0/6	N/A
AverageRank	4.522	5.778	3.700	1.761	3.822	1.417	4.256	5.478	4.261	2.283	3.578	1.144

TABLE V
THE MEAN OF THE SUCCESS RATE OVER THE 30 INDEPENDENT RUNS.

$\langle \text{wfNum}, \lambda \rangle$	Binpacking	SHWS	MGP$^-$	MGP$^+$	MCGP$^-$	MCGP$^+$
$\langle 100, 0.005 \rangle$	0.996	0.898	0.993	**0.999**	0.997	**0.999**
$\langle 200, 0.005 \rangle$	0.995	0.902	0.995	**0.999**	0.997	**0.999**
$\langle 100, 0.01 \rangle$	0.993	0.909	0.994	0.999	0.998	**1.000**
$\langle 200, 0.01 \rangle$	0.993	0.913	0.995	0.998	0.998	**0.999**
$\langle 100, 0.015 \rangle$	0.992	0.911	0.995	**0.999**	0.998	**0.999**
$\langle 200, 0.015 \rangle$	0.992	0.921	0.996	**0.999**	0.998	**0.999**
W/D/L	0/0/6	0/0/6	0/0/6	0/5/1	0/0/6	N/A
AverageRank	4.678	6.000	3.600	1.853	3.211	1.658

ous discovery of specialized heuristics that can adapt to dynamic shifts in the resource pool. Statistical "Win/Draw/Lose" (W/D/L) records show that $task$-specific MCGP significantly outperform MGP ones that evolve using a single population. These results ultimately confirm that MCGP provides a more resilient and flexible solution for managing conflicting optimization goals in high-dimensional cloud environments by integrating multitask learning for knowledge transfer and cooperative coevolution for strategy coordination.

V. CONCLUSIONS

This paper studies a coupled multitask optimization problem, the co-scheduling of online and offline workflows in the cloud, which aims to minimize the total flowtime of online workflows and the total cost of the shared cluster. To address the tripartite challenges of inherent workflow scheduling complexities, fine-grained CPU-memory configurations, and the 'solution-as-environment' bidirectional coupling, we propose multitask cooperative genetic programming to coevolve specialized subtask- and resource-selection heuristics for online and offline workflows. Extensive experiments on six scenarios based on real-world data traces demonstrate that the proposed MCGP significantly outperforms baseline algorithms in terms of total flowtime, total cost, and success rate, thereby validating the efficiency of its co-evolved $task$-specific heuristics and inter-$task$ knowledge transfer. Future research will study performance interference and vertical container elasticity via real-time resizing while integrating feature selection into the MCGP framework to simultaneously enhance resource agility, evolution efficiency, and rule interpretability.

REFERENCES

[1] F. Foundation, "State Of FinOps 2025 Report," https://data.finops.org/, accessed on 2025-10-30.

[2] Gartner, "Forecast: Public Cloud Services, Worldwide, 2022-2028, 3Q24 Update," https://www.gartner.com/en/newsroom/press-releases/2024-11-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-total-723-billion-dollars-in-2025, accessed on 2025-10-30.

[3] Flexera, "Flexera 2025 State of the Cloud Report," https://info.flexera.com/CM-REPORT-State-of-the-Cloud, accessed on 2025-10-30.

[4] C. Jiang, Y. Qiu, W. Shi, Z. Ge, J. Wang, S. Chen, C. Cerin, Z. Ren, G. Xu, and J. Lin, "Characterizing Co-Located Workloads in Alibaba Cloud Datacenters," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2381–2397, 2022.

[5] Harness, "FinOps in Focus 2025: Control Cloud Costs from Code to Production," https://www.harness.io/finops-in-focus, accessed on 2025-10-30.

[6] D. Ou, Y. Ren, and C. Jiang, "Performance Prediction Based Workload Scheduling in Co-Located Cluster," *Comput. Model. Eng. Sci.*, vol. 139, no. 2, pp. 2043–2067, 2024.

[7] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th Eur. Conf. Comput. Syst. EuroSys 2015*, 2015, pp. 1–17.

[8] L. Ye, Y. Xia, L. Yang, and C. Yan, "SHWS: Stochastic Hybrid Workflows Dynamic Scheduling in Cloud Container Services," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 3, pp. 2620–2636, 2022.

[9] J. Zhu, R. Yang, X. Sun, T. Wo, C. Hu, H. Peng, J. Xiao, A. Y. Zomaya, and J. Xu, "QoS-Aware Co-Scheduling for Distributed Long-Running Applications on Shared Clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4818–4834, 2022.

[10] J. Zhu, H. Wang, P. Su, Y. Wang, and W. Pan, "Dynamic QoS-Driven Framework for Co-Scheduling of Distributed Long-Running Applications on Shared Clusters," *IEEE Trans. Cloud Comput.*, vol. PP, no. 1, pp. 1–17, 2025.

[11] Z. Sun, B. Zhang, C. Gu, R. Xie, B. Qian, and H. Huang, "ET2FA: A Hybrid Heuristic Algorithm for Deadline-constrained Workflow Scheduling in Cloud," *IEEE Trans. Serv. Comput.*, vol. 16, no. 3, pp. 1807–1821, 2023.

[12] A. Gupta, Y. S. Ong, and L. Feng, "Multifactorial Evolution: Toward Evolutionary Multitasking," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 343–357, 2016.

[13] L. Miguel Antonio and C. A. Coello Coello, "Coevolutionary Multiobjective Evolutionary Algorithms: Survey of the State-of-the-Art," *IEEE Trans. Evol. Comput.*, vol. 22, no. 6, pp. 851–865, 2018.

[14] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated Design of Production Scheduling Heuristics: A Review," *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 110–124, 2016.

[15] X. Ma, A. Zhou, S. Zhang, Q. Li, A. X. Liu, and S. Wang, "Dynamic Task Scheduling in Cloud-Assisted Mobile Edge Computing," *IEEE Trans. Mob. Comput.*, vol. 22, no. 4, pp. 2116–2130, 2023.

[16] Z. Sun, F. Zhang, Y. Mei, H. Huang, C. Gu, B. Wang, and M. Zhang, "Cooperative Coevolution Genetic Programming for Dynamic Joint Workflow Scheduling and Container Scaling in Cloud-Fog Computing," *IEEE Trans. Serv. Comput.*, vol. 19, no. 1, pp. 225–239, 2026.

[17] Z. Sun, Y. Mei, F. Zhang, H. Huang, C. Gu, and M. Zhang, "Multi-Tree Genetic Programming Hyper-Heuristic for Dynamic Flexible Workflow Scheduling in Multi-Clouds," *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2687–2703, 2024.

[18] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.