Cooperative Coevolution Genetic Programming for Dynamic Joint Workflow Scheduling and Container Scaling in Cloud-Fog Computing

Zaixing Sun, Fangfang Zhang, *Member, IEEE*, Yi Mei, *Senior Member, IEEE*, Hejiao Huang, Chonglin Gu, Bin Wang, and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Cloud-Fog computing has emerged as an essential paradigm to support the growing demand for real-time data processing driven by the Internet of Things. By integrating the extensive computing capabilities of cloud data centres with the low-latency benefits of fog nodes, this architecture increases resource utilisation and improves quality of service. However, the dynamic and heterogeneous nature of cloud fog environments poses significant workflow scheduling challenges, especially when optimising multiple trade-offs such as latency, cost, energy consumption, and resource utilisation. This paper investigates the many-objective dynamic workflow scheduling problem under deadline constraints in container-based cloud-fog computing environments (MDWS-CoCF). Unlike existing studies that primarily focus on horizontal scaling, this work considers both vertical and horizontal scaling of containers, allowing for real-time adjustments of container configurations based on task-specific requirements. To address this complex problem, we first develop a dynamic workflow scheduling simulator that models real-world scenarios, including a variety of task categories and container scalability. Based on this simulator, we propose a Cooperative Coevolution Genetic Programming (CCGP) approach that evolves specialised heuristics for task selection, resource allocation, and container deployment to facilitate adaptive and efficient scheduling in MDWS-CoCF. Extensive simulations using real-world data traces show that the proposed CCGP approach significantly outperforms existing baseline algorithms, achieving superior performance as measured by the HyperVolume and Inverted Generational Distance metrics. The results show that the evolved heuristics are robust and effective under different dynamic scenarios, ensuring balanced optimisation of many objectives.**

Received 21 March 2025; revised 23 October 2025; accepted 15 November 2025. Date of publication 24 November 2025; date of current version 5 February 2026. This work was supported in part by the Major Key Project of PCL, China, under Grant PCL2025A13 and in part by Shenzhen Science and Technology Program under Grant GXWD20220817124827001 and Grant SYSPG2024 1211173609009. *(Corresponding authors: Chonglin Gu; Bin Wang.)*

Zaixing Sun and Bin Wang are with the Department of New Networks, Pengcheng Laboratory, Shenzhen 518000, China (e-mail: zaixing.sun@micc.hitsz.edu.cn; wangb02@pcl.ac.cn).

Fangfang Zhang, Yi Mei, and Mengjie Zhang are with the Evolutionary Computation and Machine Learning Research Group, Centre for Data Science and Artificial Intelligence, School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: fangfang.zhang@ecs.vuw.ac.nz; yi.mei@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Hejiao Huang and Chonglin Gu are with the Shenzhen Key Laboratory of Internet Information Collaboration, School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen 518055, China (e-mail: huanghejiao@hit.edu.cn; guchonglin@hit.edu.cn).

Digital Object Identifier 10.1109/TSC.2025.3636121

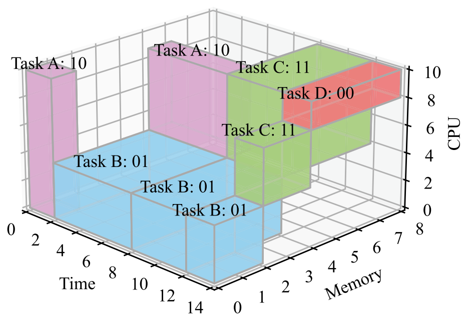
***Index Terms*—Cloud computing, fog computing, workflow scheduling, dynamic many-objective optimisation, genetic programming hyper-heuristic, elastic scaling.**

I. Introduction

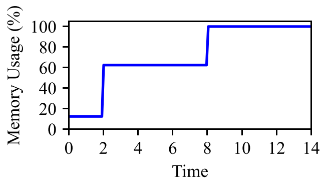
CLOUD-FOG computing has been brought to the forefront of modern computing paradigms by the rapid growth of the Internet of Things (IoT) and the increasing demand for real-time data processing. This hybrid architecture leverages the massive computing power of cloud data centres with the localised, latency-reducing capabilities of fog nodes located closer to data sources. Integrating these two tiers aims to optimise resource utilisation, reduce latency and improve quality of service (QoS) [1], [2]. However, the dynamic and heterogeneous nature of cloud-fog environments presents significant challenges, particularly in dynamic workflow scheduling, which requires adaptive strategies to address multiple, often conflicting objectives such as latency, cost, and energy consumption [3], [4].

In recent years, container-based virtualisation is emerging as a crucial technology in cloud-fog computing environments, providing lightweight, scalable, and efficient resource management capabilities. Containers encapsulate applications and their dependencies, ensuring consistent execution across different environments and enhancing resource utilisation [5], [6]. A key advantage of containers is their inherent horizontal and vertical elasticity. *Horizontal scaling* leverages their fast startup time to deploy multiple containers simultaneously, while *vertical scaling* allows real-time adjustment of container configurations, such as CPU and memory [7]. For example, Kubernetes supports in-place resizing of CPU and memory resources without the need to restart the container, thereby improving responsiveness to dynamic workloads [8]. Similarly, Docker offers runtime options to set memory and CPU limits for containers, facilitating efficient resource management [9].

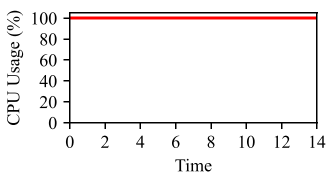
However, existing research on task scheduling primarily focuses on horizontal scaling, with few studies on vertical scaling. Although several studies [10], [11], [12] have addressed workflow scheduling in containerised environments, their models usually assume that containers have static resource configurations that are either inherited from or equivalent to those of virtual machines (VMs). While these models benefit from faster container startups than VMs, they overlook the vertical elasticity



(a) The execution of multiple containers on a VM over time, along with their CPU and memory occupancy.



(b) The memory usage of the VM over time.



(c) The CPU usage of VM over time.

Fig. 1. A simple example of containers with vertical scaling support running on a single VM. '10' indicates that the task adjusts CPU only, '01' indicates that the task adjusts memory only, '11' indicates that the task adjusts both CPU and memory, and '00' indicates that the task does not adjust any configurations.

of containers. Therefore, to cover the important aspect of vertical elasticity not yet considered in the literature, this paper will not only consider the horizontal elasticity of containers, but also their vertical elasticity along with different task categories.

Existing research tends to overlook the vertical elasticity of containers and the diversity of task categories in real-world scenarios. These task categories include (1) computation/CPU-intensive tasks that require high CPU utilisation, such as data analysis, image/video processing, and machine learning algorithms; (2) memory/data-intensive tasks that require high memory utilisation, such as data aggregation, caching, and data preprocessing; (3) balanced tasks that require a mix of CPU and memory resources, such as web servers and database management systems; and (4) communication-intensive tasks that involve significant data transfer and require efficient network communication, such as real-time streaming applications. Most existing studies mainly focus on CPU-intensive tasks [10], [13], [14], [15], which are affected by the computational capacity of the CPU, with tasks having shorter execution times when executed on resources with high computational capacity, and vice versa. Few studies simultaneously consider these task categories. By supporting vertical scaling, multiple containers can run on the same VM (or host) at the same time, which can not only avoid switching multiple VMs to save energy consumption or cost, but also improve resource utilisation by adjusting the configuration of containers. Fig. 1(a) is a simple example of containers with vertical scaling support running on a single VM. Fig. 1(b) and (c) show the CPU and memory usage of the VM over time, respectively. Containers share the same VM by adjusting the configuration of CPU or memory.

Given these gaps in the literature, this paper focuses on the Many-objective Dynamic Workflow Scheduling under deadline constraints in Container-based Cloud-Fog computing (MDWS-CoCF). In MDWS-CoCF, workflows have different priority weights or sensitivity to deadline violations, and tasks of various categories are executed in containers with adjustable configurations, such as CPU and memory, based on task requirements. The objectives are to minimise the workflow-oriented total weighted tardiness, the cloud-oriented total monetary cost, the fog-oriented total energy consumption, and maximise the system-oriented total resource utilisation. Previous studies have covered different aspects of workflow (task) scheduling. For example, Xu et al. [16] studied dynamic workflow scheduling in fog environments, while Azizi et al. [17] studied deadline-aware and energy-efficient static independent task scheduling, treating all computing resources as fog nodes. Tan et al. [18] focused on online resource allocation in container-based clouds, but unlike this study, they did not consider containers that are dynamically released and reconfigured based on task execution. These works primarily considered CPU-intensive tasks and did not address the full spectrum of task categories or the dynamic reconfiguration of container resources.

Therefore, the MDWS-CoCF problem inherits many challenges of traditional workflow (task) scheduling in cloud or fog environments and container-based resource allocation, such as deadline constraints, determining the scheduling sequence of workflows (tasks), and handling the trade-offs between capacity and cost in resource selection [11], [19], [20]. Additionally, MDWS-CoCF introduces new challenges, including dynamically adjusting container configurations in real-time based on changing system states, which involves identifying which containers to adjust and how to adjust them to optimise resource utilisation without disrupting ongoing tasks. Furthermore, balancing many conflicting objectives such as tardiness, cost, energy consumption, and resource utilisation simultaneously adds complexity to the scheduling process. Ensuring the scalability and responsiveness of the scheduling framework to handle an increasing number of tasks while maintaining real-time performance and adapting to varying workloads requires a sophisticated and robust approach.

To overcome these challenges, we aim to develop a new dynamic workflow scheduling simulator, which schedules sequentially according to the event queue and employs scheduling heuristics to make decisions at each decision point to make the scheduling process continue. MDWS-CoCF involves three main decisions, i.e., the *task selection decision* to select a ready task, the *resource selection decision* to select a resource in the cloud-fog system to execute the ready task, and the *container deployment decision* to assign an appropriate configuration to the container (re)assignment to execute the task. The first two decisions, task and resource selection decisions, are already available in traditional workflow scheduling [19], [20], [21], [22], [23], [24], while container deployment decision is purposefully introduced in this paper to support vertical scaling of containers. We propose a novel Cooperative Coevolution Genetic Programming (CCGP) approach to automatically generate, select, and evolve scheduling heuristics for these decisions based on the state characteristics of the system.

Several reasons motivate us to propose the CCGP approach. *First*, *Genetic Programming Hyper-Heuristic* (GPHH) [25], [26] is an automated methodology for evolving heuristics to solve many combinatorial optimisation problems, including workflow scheduling [16], [21], [27], job shop scheduling [28], [29], and routing problems [30], [31]. GPHH can mitigate the limitations of heuristic approaches, such as greediness and dependence on expert domain knowledge [16], [32], as well as the drawbacks of meta-heuristics, such as long scheduling delays caused by iterative optimisation. The goal of GPHH is to explore the "heuristic search space" of the problems instead of the solution search space in the cases of heuristics and meta-heuristics. In other words, the output of GPHH is a set of rules or heuristics/policies instead of a schedule or a solution, which is typically the case for genetic algorithm or other heuristic methods. The best heuristic discovered by GPHH during training can make decisions in dynamic/real-time workflow scheduling. *Second*, Cooperative Coevolution is a reciprocal evolutionary exchange between interacting populations [33]. Cooperative coevolution can help maintain search diversity by decomposing problems into sub-components, and when integrated with multi-objective optimization techniques, it can contribute to finding well-distributed solutions along the Pareto front. *Third*, CCGP leverages the combined strengths of GPHH and Cooperative Coevolution to address the dynamic and heterogeneous nature of cloud-fog environments. By decomposing the complex scheduling problem into sub-problems, CCGP evolves specialised heuristics for task selection, resource selection, and container deployment. This approach allows CCGP to adapt to real-time changes in system states and workload patterns, ensuring robust and efficient scheduling under varying conditions. Therefore, CCGP is a promising technique for our problem.

To the best of our knowledge, this is the first work on MDWS-CoCF that supports vertical and horizontal scaling of containers. The contributions of this paper are as follows:

- We provide a constrained optimisation model for the many-objective dynamic workflows scheduling under deadline constraints in container-based cloud-fog computing, which covers the important aspect of vertical and horizontal scaling of containers not yet considered in the literature.
- We customise a new dynamic workflow scheduling simulator to mimic the scheduling process in real-world scenarios as the existing simulator does not support multiple task categories and vertical and horizontal scaling of containers. The scheduling heuristic is used to make a decision at each decision point to obtain the final schedules.
- We propose a CCGP approach, which is comprised of three rules, i.e., the *task selection rule* to select a ready task, the *resource selection rule* to select a resource in the cloud-fog system to execute the ready task, and the *container deployment rule* to assign an appropriate configuration to the container (re)assignment to execute the task. We design a number of terminals based on the scheduling process, which can clearly describe the state of the cloud-fog system.
- Extensive simulation experiments using two real-world data traces, the Cluster-trace-v2018 from Alibaba Cloud and scientific workflows from the Pegasus project, demonstrate that the proposed algorithm significantly outperforms existing baselines. Additionally, the scheduling heuristics evolved through CCGP effectively combine and interact features, ensuring robustness and adaptability across diverse dynamic scenarios and requirements.

Due to the space limitation, the section of Related Work is given in the Supplementary Materials.

II. Container-Based Cloud-Fog Workflow Scheduling Modelling

A. Container-Based Cloud-Fog Computing System Architecture

We consider a standard distributed and heterogeneous cloud-fog computing environment as shown in Fig. 2.

Real-time Applications from IoT Devices: IoT devices generate a continuous stream of data, often requiring real-time processing and analysis. These devices may submit multiple workflow applications simultaneously. The workflows vary in complexity and resource requirements, demanding a flexible and responsive scheduling mechanism to ensure timely and efficient execution. IoT devices can be smart wearables, automobile sensors, security sensors, smart home appliances, thermostats, industry devices, and so on. Most IoT devices are resource constrained. They suffer from limited resources including processing capabilities, memory, and battery power. Therefore, they offload their tasks to the edge or cloud layers.

Broker or Resource Management System: The broker or resource management system is the core of the architecture. It consists of a *resource monitor* that monitors the status of tasks and resources, and a *workflow scheduler* that generates task scheduling and resource deployment decisions based on the resource monitor. Through the scheduling decisions, the *container orchestrator* encapsulates the tasks; the *deployment plan* deploys the tasks to the specified resources; the *elastic scaling controller* adjusts the container's resource configuration to achieve vertical scaling, and starts or shuts down resources to achieve horizontal scaling.

Container-based Cloud-Fog Computing: The infrastructure consists of a number of heterogeneous computing resources with different configurations that form the foundation of the system, including fog nodes and cloud servers: *Fog Nodes*: Deployed at the network edge, close to the data sources, fog nodes are responsible for handling latency-sensitive and real-time tasks. These nodes, such as high-end servers, set-top boxes, Raspberry Pis, routers, and personal computers, possess certain computing and storage capabilities, enabling them to respond quickly to local device requests due to their proximity to terminal IoT devices [17]. *Cloud Servers*: Composed mainly of a set of VMs with high computing power and storage capacity, cloud servers handle more complex and computation-intensive tasks. Despite having much higher communication latency due to multiple hops from users, cloud servers can centrally manage large-scale data and applications, providing efficient resource scheduling and data analysis services.

IoT devices suffer from low bandwidth and high latency when communicating with cloud servers because IoT devices are connected to cloud servers via Wide Area Network (WAN), which provides low bandwidth, and the distance of cloud servers and IoT devices which leads to high latency [34]. However,

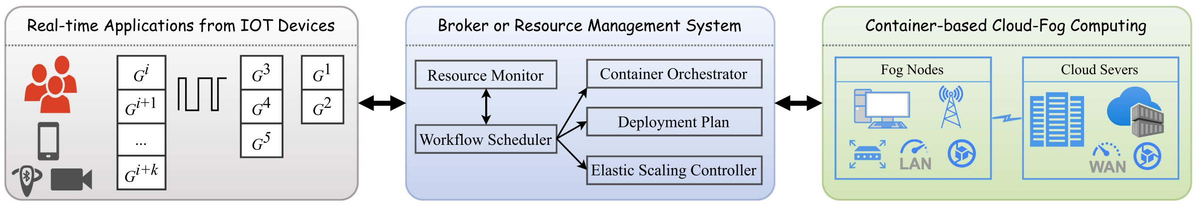


Fig. 2. The dynamic workflow scheduling architecture in container-based cloud-fog computing.

Cloud Severs provide an unlimited number of highly configurable resources on a pay-as-you-go basis. *Fog Nodes* have fewer resources (e.g., CPU, RAM) in comparison to cloud servers, while they provide higher bandwidth with less latency for IoT devices since they can be accessed via Local Area Network (LAN) [34], [35]. We assume full connectivity between all computing nodes, regardless of whether they are fog or cloud nodes. *Fog Nodes* provide a fixed number of resources.

Tasks are packaged as containers through existing image files, and then the containers are deployed to the resource. A resource can deploy and run multiple containers simultaneously, but cannot exceed the configurations of the resource. Containers are minimal units, i.e. a container can only run one task at a time, and the CPU and memory of a container can be adjusted to increase or decrease its capacity. The execution time of a task is affected by both the CPU and memory configurations of the container.

B. Problem Formulation

We consider a computing resource in the cloud-fog environment to be a host. Tasks 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 task at a time, and the CPU and memory of a container can be adjusted to increase or decrease its capacity. The execution time of a task is affected by both the CPU and memory configurations of the container and is divided into the following four categories. Let $\phi(\alpha)$ be the category of a task α.

- $\phi(\alpha) = 10$. The container's memory needs to meet the memory demand of the task, while improving the container's CPU configuration can reduce the execution time of the task, and vice versa. Such tasks are usually computation-intensive.
- $\phi(\alpha) = 01$. The container's CPU needs to meet the CPU demand of the task, while improving the container's memory configuration can reduce the execution time of the task, and vice versa. Such tasks are usually memory-intensive.
- $\phi(\alpha) = 11$. Improving both the container's memory and CPU configurations can reduce the execution time of the task, and vice versa. Such tasks are usually balanced between computation and memory.
- $\phi(\alpha) = 00$. Both the container's CPU and memory meet the task's requirements, and improving the container's configuration can't reduce the task's execution time. Such tasks are usually communication-intensive.

Let $\mathcal{R}$ be the set of heterogeneous computing resources. Each resource r is associated with a type $\theta(r) \in \Theta = \mathcal{C} \cup \mathcal{F}$, the size of CPU $\Gamma^c(r)$, the size of memory $\Gamma^m(r)$. $\mathcal{C}$ and $\mathcal{F}$ are the set of resource types provided by the *Cloud Severs* and *Fog Nodes*, respectively. Let B_{WAN} and B_{LAN} be the bandwidth of the LAN and WAN respectively. Let L_{WAN} and L_{LAN} be the latency of the LAN and WAN respectively [34]. There are a per-unit lease price $c^e(r)$ and a price of transmitting unit data $c^d(r)$ if the resource r belongs to *Cloud Severs*. If this resource belongs to *Fog Nodes*, there are dynamic power and static power. 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 $A(g)$. The arrival time, weight, and deadline of workflow g are $\rho^a(g)$, $\rho^w(g)$, and $\rho^d(g)$, respectively. Each task $\alpha \in A(g)$ is represented by a tuple $\{\phi(\alpha), \tau^l(\alpha), \mu^c(\alpha), \ \mu^m(\alpha), A^{pre}(\alpha), A^{suc}(\alpha)\}$, where $\phi(\alpha) \in \Phi = \{00, 01, 10, \ 11\}$ is a binary variable and denotes the category of task α, $\tau^l(\alpha)$ is the computational workload of the task, $\mu^c(\alpha)$ is minimum computational resource requirement, $\mu^m(\alpha)$ is minimum memory resource requirement, $A^{pre}(\alpha) \subseteq A(g)$ is a set of predecessor tasks, and $A^{suc}(\alpha) \subseteq A(g)$ 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 α_2 $\in A^{suc}(\alpha_1)$.

We formulate the deadline-constrained dynamic joint workflow scheduling and container scaling problem as an optimisation problem aiming to minimise the fog-oriented total energy consumption, cloud-oriented total monetary cost, and workflow-oriented total weighted tardiness, and to maximise resource utilisation, which subject to the following constraints:

- No task can be executed until all its predecessor tasks have been completed.
- Each task must be executed by a container whose configuration can be changed.
- 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.

We assume scheduling as a discrete-time control problem as standard in previous work [18], [36], [37], [38]. 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 a workflow is submitted, a task starts execution, or a container configuration is updated. 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 $\nu^c(\alpha, t)$ and $\nu^m(\alpha, t)$ be the sizes of the CPU and memory allocated to the container used to execute the task α at time t.

The problem can then be formulated as

$$\min\ \mathbb{E} = \sum_{r\in\mathcal{R}} \int_T P(r,t)dt, \tag{1}$$

$$\min\ \mathbb{C} = \sum_{r\in\mathcal{R}} \int_T C^e(r,t)dt + \sum_{g\in\mathcal{G}} \sum_{\substack{\alpha_1\in A(g),\\ \alpha_2\in A^{suc}(\alpha_1)}} C^d(\alpha_1,\alpha_2), \tag{2}$$

$$\min\ \mathbb{T} = \sum_{\forall\ g\in\mathcal{G}} \rho^w(g)\times \max\left\{0, \max_{\alpha\in A(g)}\{x^f(\alpha)\} - \rho^d(g)\right\}, \tag{3}$$

$$\max\ \mathbb{U} = \frac{\sum_{r\in\mathcal{R}} \int_T \left(U^c(r,t)+U^m(r,t)\right)dt}{2\times|T|\times|\mathcal{R}|}, \tag{4}$$

$$\text{s.t.}:\ x^s(\alpha) \geq \rho^a(g),\ \forall\ g\in\mathcal{G},\ \alpha\in A(g), \tag{5}$$

$$x^f(\alpha_1)+x^c(\alpha_1,\alpha_2)+x^l(\alpha_2) \leq x^s(\alpha_2), \forall\alpha_2\in A^{suc}(\alpha_1), \tag{6}$$

$$\begin{cases} \int_{x^s(\alpha)}^{x^f(\alpha)} \frac{1}{\mu^c(\alpha)\times\mu^m(\alpha)}\,dt = \tau^l(\alpha), & \phi(\alpha)=00,\\ \int_{x^s(\alpha)}^{x^f(\alpha)} \frac{1}{\mu^c(\alpha)\times\nu^m(\alpha,t)}\,dt = \tau^l(\alpha), & \phi(\alpha)=01,\\ \int_{x^s(\alpha)}^{x^f(\alpha)} \frac{1}{\nu^c(\alpha,t)\times\mu^m(\alpha)}\,dt = \tau^l(\alpha), & \phi(\alpha)=10,\\ \int_{x^s(\alpha)}^{x^f(\alpha)} \frac{1}{\nu^c(\alpha,t)\times\nu^m(\alpha,t)}\,dt = \tau^l(\alpha), & \phi(\alpha)=11. \end{cases} \tag{7}$$

$$\begin{cases} \nu^c(\alpha,t) \geq \mu^c(\alpha), & \forall t\in T,\\ \nu^m(\alpha,t) \geq \mu^m(\alpha), & \forall t\in T. \end{cases} \tag{8}$$

$$\begin{cases} \sum_{\forall y(\alpha)=r} \nu^c(\alpha,t) \leq \Gamma^c(r), & \forall r\in\mathcal{R}, \forall t\in T,\\ \sum_{\forall y(\alpha)=r} \nu^m(\alpha,t) \leq \Gamma^m(r), & \forall r\in\mathcal{R}, \forall t\in T. \end{cases} \tag{9}$$

$$x^l(\alpha) = \begin{cases} L_{LAN}, & y(\alpha)\in\mathcal{F},\\ L_{WAN}, & y(\alpha)\in\mathcal{C}. \end{cases} \tag{10}$$

$$x^c(\alpha_1,\alpha_2) = \begin{cases} 0, & y(\alpha_1)=y(\alpha_2),\\ \frac{d(\alpha_1,\alpha_2)}{B_{LAN}}, & y(\alpha_1)\neq y(\alpha_2),\ y(\alpha_1),y(\alpha_2)\in\mathcal{F},\\ \frac{d(\alpha_1,\alpha_2)}{B_{WAN}}, & \text{otherwise}, \end{cases} \tag{11}$$

$$U^c(r,t) = \frac{\sum_{\forall y(\alpha)=r} \nu^c(\alpha,t)}{\Gamma^c(r)}, \tag{12}$$

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

$$P(r,t) = \begin{cases} p^s(r)+p^{dc}(r)\times U^c(r,t) + p^{dm}(r)\times U^m(r,t), & \text{if } r\in\mathcal{F},\\ 0, & \text{otherwise}, \end{cases} \tag{14}$$

$$C^e(r,t) = \begin{cases} c^e(r), & \text{if } r\in C, U^c(r,t)\neq 0,\\ 0, & \text{otherwise}, \end{cases} \tag{15}$$

$$C^d(\alpha_1,\alpha_2) = \begin{cases} c^d(r)\times d(\alpha_1,\alpha_2), & \text{if } y(\alpha_1)\in C, y(\alpha_2)\in\mathcal{F},\\ 0, & \text{otherwise}, \end{cases} \tag{16}$$

where $\forall\ g\in\mathcal{G}$, $\forall\ \alpha,\alpha_1,\alpha_2\in A(g)$, $\forall\ r\in\mathcal{R}$, and $\forall t\in T$. Eqs. (1), (2), (3), and (4) are the four considered objectives, respectively. Eq. (1) is the fog-oriented total energy consumption, which is the sum over all fog nodes of the time integral of the instantaneous power $P(r,t)$ for each node. Eq. (2) is the cloud-oriented total monetary cost, which includes the cost of leasing cloud severs based on pay-as-you-go and the communication cost. Eq. (3) is the workflow-oriented total weighted tardiness. Eq. (4) is the total resource utilisation of system, which is the average combined utilisation of CPU and memory across all resources over the entire scheduling period. Constraint (5) ensures that the workflow cannot be executed until it is submitted. In other words, the constraint is also regarded as the fact that any information about the workflow is unknown to the system until it is submitted. Constraint (6) 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. Eq. (7) governs the execution of different categories of tasks, ensuring that each task α completes its computational workload $\tau^l(\alpha)$ within its time slot $[x^s(\alpha), x^f(\alpha)]$ under variable container configurations. Constraint (8) indicates that the configuration of the container needs to satisfy the requirements of the task. Constraint (9) indicates that the sum of the CPU and memory capacities of the containers allocated on the resource cannot exceed the corresponding resource's capacity at time t. Constraint (10) defines the network latency $x^l(\alpha)$ associated with executing task α, which depends on whether it is scheduled on a fog node or a cloud node. Eq. (11) is the communication time between each task α_1 and its successor task α_2. Eqs. (12) and (13) are the CPU-based resource utilisation and memory-based resource utilisation, respectively. Eq. (14) models the instantaneous power consumption $P(r,t)$ of a resource r at time t. The power is composed of a static power $p^s(r)$ and a dynamic power that scales linearly with both CPU and memory utilisation. The power consumption of non-fog resources (e.g., cloud servers) is considered zero in this context, as their energy cost is managed by the cloud provider. Eq. (15) indicates the resources in the cloud server will be charged when running the task. Eq. (16) indicates that data will be charged when it is transmitted from the cloud server.

III. Dynamic Workflow Scheduling Simulator

Given that existing simulation platforms cannot support MDWS-CoCF, especially in terms of simultaneous consideration of multiple task categories, and horizontal and vertical scaling of containers. We develop a new simulator to imitate the scheduling process. This simulator involves three key decisions and is driven by discrete events. We will first introduce these three decisions, followed by a detailed explanation of each event. The following three rules are employed to make the corresponding decisions during the scheduling process:

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.

Resource selection rule (Rule 2): to select a resource in the cloud-fog system to execute the selected task. There are three main steps. First of all, eliminating the resources that con not meet the minimum requirements of CPU or Memory of the task. Secondly, it filters out the resources that cannot meet the sub-deadline to execute the task. If no resource remains, then the resource in the cloud with the earliest finish time is selected to execute the task from all available resources. Otherwise, the resource selection rule is used to select the resource. In cases where multiple resources have the same priority, we prioritise selecting an existing resource that does not need to be created.

Container deployment rule (Rule 3): to assign the appropriate configuration (CPU and memory) for containers to perform tasks. First of all, we screen out the containers that need to adjust CPU and memory respectively; under the premise of satisfying the minimum required execution of all containers, we calculate the remaining allocable CPU and memory of the resource. Then, the remaining resources are allocated to these containers. If there is a container where CPU and Memory need to be adjusted at the same time, CPU and Memory are allocated in turn according to the percentage of remaining capacity.

According to the allocable capacity and the number of container requirements, CPU and memory are allocated and the same strategy is adopted. The following three schemes can be adopted to assign the appropriate configuration: (i) average allocation; (ii) randomly generated allocation scheme and random allocation; (iii) randomly generated allocation scheme and allocation according to the priority of the containers in descending order. Among them, the above three schemes are selected by roulette according to a given probability sequence. To learn to choose an allocation scheme matching the container state, we set the probability to [0.2,0.2,0.6].

The discrete event-driven simulation process with the three rules maintains an event queue $\mathcal{E}$ sorted by trigger time. Each event $\mathcal{E}_e$ is associated with the event name $\mathcal{E}_e^N$, trigger time $\mathcal{E}_e^T$, workflow involved $\mathcal{E}_e^W$, task involved $\mathcal{E}_e^A$, service resource involved $\mathcal{E}_e^S$, and container involved $\mathcal{E}_e^C$. Fig. 3 shows the flowchart of dynamic event triggering during a workflow's lifecycle, from submission to completion. **Events** represent distinct state transitions, each with unique trigger conditions and behaviors, that form the backbone of the simulation engine. **Rules** represent the decision-making logic that our Genetic Programming approach evolves and evaluates at these transitions. This decoupling is essential as it allows the scheduling rules to be modified and tested independently of the core simulation engine, providing crucial modularity and extensibility. According to the event name, we introduce (1) *when* each event is triggered and (2) *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 a resource to execute the task according to the *resource selection rule*. Once the resource that will execute the task has been determined, we can determine the start execution time of the task from the terminals associated with that resource. We add the task to the wait queue of the resource that will execute it.
- `StartExecuteTask`. *When:* after the ready task has received data from all predecessors and the assigned resource is active. *What:* remove the task from the wait queue and add it to the run queue. We reconfigure running containers based on *container deployment rule*.
- `CompleteExecuteTask`. *When:* after the execution of task is finished. *What:* remove the task from the run queue and add its child task to `TaskReady` event queue if its child task is ready. We reconfigure running containers based on *container deployment rule*.
- `CompleteExecuteWorkflow`. *When:* after all the tasks in a workflow have been finished. *What:* record that the workflow has been scheduled.

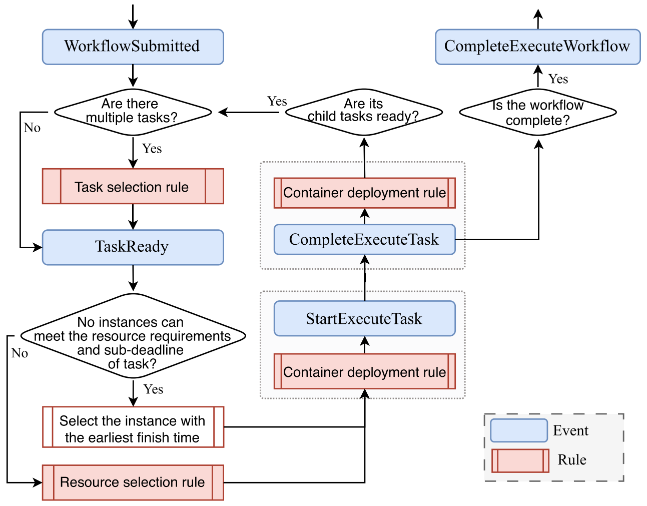

WorkflowSubmitted
Are there multiple tasks?
No
Yes
Task selection rule
TaskReady
No instances can meet the resource requirements and sub-deadline of task?
No
Yes
Select the instance with the earliest finish time
Resource selection rule
Are its child tasks ready?
Yes
Container deployment rule
CompleteExecuteTask
StartExecuteTask
Container deployment rule
CompleteExecuteWorkflow
Yes
Is the workflow complete?
Event
Rule

Fig. 3. The flowchart of dynamic event triggering during a workflow's lifecycle, from submission to completion.

The *container deployment rule* is triggered twice in the flowchart (Fig. 3): first, to configure a container before a new task starts, and second, to vertically scale the container after a task completes. This dual invocation is necessary to adapt to changing resource demands and improve overall utilisation.

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.

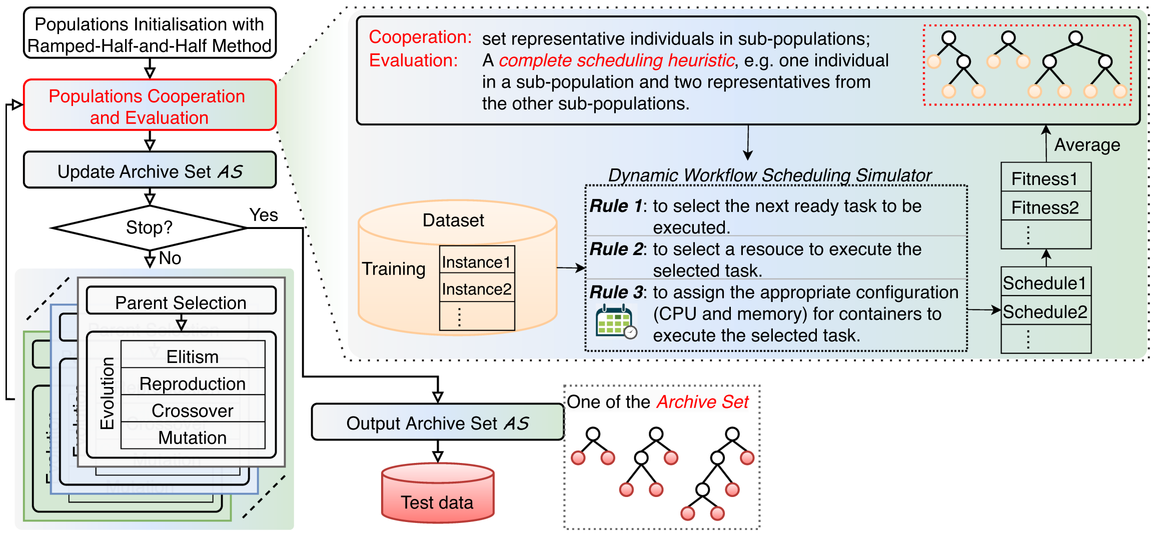

Populations Initialisation with Ramped-Half-and-Half Method
Populations Cooperation and Evaluation
Update Archive Set AS
Stop?
Yes
No
Parent Selection
Evolution
Elitism
Reproduction
Crossover
Mutation
Output Archive Set AS
Test data
One of the Archive Set
Cooperation: set representative individuals in sub-populations;
Evaluation: A complete scheduling heuristic, e.g. one individual in a sub-population and two representatives from the other sub-populations.
Dynamic Workflow Scheduling Simulator
Rule 1: to select the next ready task to be executed.
Rule 2: to select a resouce to execute the selected task.
Rule 3: to assign the appropriate configuration (CPU and memory) for containers to execute the selected task.
Dataset
Training
Instance1
Instance2
Average
Fitness1
Fitness2
Schedule1
Schedule2

Fig. 4. The flowchart of the CCGP approach.

IV. The Proposed Dynamic Workflow Scheduling Algorithm

A. Cooperative Coevolution Genetic Programming

We propose the CCGP approach to automatically generate the above tree-based rules to make decisions. The training process is performed offline and the trained rules are deployed to make online scheduling decisions. The flowchart of the CCGP approach is shown in Fig. 4, where three populations of rules are evolved through the following evolutionary mechanisms.

Populations Initialisation with Ramped-Half-and-Half Method: Initially, individuals in the three populations are generated using the ramped-half-and-half method [39], where terminals and functions are randomly selected and combined. Specifically, half of the individuals in a population are initialised with the maximum depth set in advance, while the remaining individuals in the population are initialised 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 *Dynamic Workflow Scheduling Simulator* in Section III. After fitness evaluation, each individual has a fitness that represents its quality. We employ cooperative coevolution to evolve three decision rules in three subpopulations [21], [40]. Each subpopulation corresponds to one rule type: task selection, resource selection, or container deployment. In each generation, a representative individual is first selected from each subpopulation. Each individual within a subpopulation then cooperates with these representatives from the other two subpopulations to form a complete scheduling heuristic for making three 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 employ a Pareto-based selection strategy [41], i.e., we first do a *non-dominated sorting of the individuals within each population*, and then randomly select a representative from the first front (i.e., with the lowest rank).

Update Archive Set $\mathcal{AS}$: When updating the archive set, we record all complete scheduling heuristics represented by all individuals in the populations. After removing the duplicate solutions, the first-front non-dominated solutions are copied into the new archive set $\mathcal{AS}$ based on the non-dominated sorting and crowding distance methods [41]. This ensures that each solution of the output archive set is a complete scheduling heuristic for solving the problem.

Parent Selection: If the stopping criterion is met, the archive set $\mathcal{AS}$ so far is considered as the best evolved scheduling heuristics for MDWS-CoCF. Otherwise, the populations will be updated evolutionarily in turn. The NSGA-III differs significantly from NSGA-II in its selection mechanism, which is designed to improve the results of many-objective problems with 3 to 15 objectives [42], [43]. Therefore, we employ the reference-points-based selection mechanism in NSGA-III to select parents for generating offspring. Following [42], [43], we predefine 15 reference points that are uniformly distributed in the objectives space. Each solution is associated with the nearest reference point by perpendicular distance, and individuals are finally selected using a niche-preservation technique to improve population diversity. More details can be found in [42], [43].

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. *Elitism*: A predefined number of elite individuals are directly copied into the next generation. These elites are the top individuals picked up from the current population based on the above non-dominated sorting and crowding distance methods, ensuring the preservation of the best-performing solutions. *Reproduction* selects individuals by tournament selection, and copies them into the next generation. *Crossover*: This operator creates a new offspring by swapping

TABLE I
TERMINAL AND FUNCTION SETS OF CCGP

No.	Nation	Description	Rule 1	Rule 2	Rule 3
1	CPUA	The minimum CPU requirement of a task.	✓		
2	MEMA	The minimum memory requirement of a task.	✓		
3	CAT	The category of a task.	✓		
4	ET	The execution time of a task on a reference resource.	✓		
5	SD	The sub-deadline of a task.	✓		
6	UR	The upward rank of a task.	✓		
7	VCT	The average communication time for a task.	✓		
8	NKQ	The number of tasks in ready queue.	✓		
9	NRA	The number of remaining (or unscheduled) tasks in a workflow.	✓		
10	WT	The weight of a workflow.	✓		
11	TETQ	The total ET of tasks in ready queue.	✓		
12	TETRA	The total ET of remaining tasks in a workflow.	✓		
13	NPA	The number of direct predecessors for a task.	✓		
14	NSA	The number of direct successors for a task.	✓		✓
15	CPUR	The CPU size of a resource.		✓	
16	MEMR	The memory size of a resource.		✓	
17	PR	The unit price of a resource.		✓	
18	STPR	The static power of a resource.		✓	
19	CTR	The communication time of a task on a resource.		✓	
20	ETAR	The minimum execution time of a task on a resource.		✓	
21	CRCPU	The current remaining CPU of a resource.		✓	
22	CRMEM	The current remaining memory of a resource.		✓	
23	FAAT	The factor of task category on a resource.		✓	
24	AAT	The actual available time of a resource.		✓	
25	CC	The communication cost of a task while it is executed on a resource.		✓	
26	NKW	The number of the scheduled tasks belonging to the same workflow in cloud or fog.		✓	
27	ST	The slack time of task, which is $(AAT-SD)$.		✓	✓
28	RLA	The remaining workload of the task executed by the container.			✓
29	PACC	The previously allocated configuration for the container and initially defaults to 0.			✓
30	MICC	The minimum configuration of the container, i.e. the minimum requirement for the task.			✓
Function set		+, -, *, protected /, Max, Min	✓	✓	✓

subtrees between two parents. First, a subtree is randomly selected from a parent. Then, an individual is selected from the archive set $\mathcal{AS}$, and one of its subtrees is randomly chosen. Finally, we replace the parent's subtree with the selected subtree to form a new offspring. *Mutation*: This operator introduces random variations by modifying a single parent. A new subtree is randomly generated using the same grammar rules that govern the initial population creation. This new subtree then replaces a randomly selected subtree within the parent, resulting in a mutated offspring.

B. Representation, Terminal Set, and Function Set

To evolve a scheduling heuristic with three rules for the MDWS-CoCF problem, we employ three tree-based individuals represent task selection rule, resource selection rule and container deployment rule, respectively. We design 30 terminals based on the scheduling process, which indicate the characteristics related to workflows, tasks, resources and containers, to describe the state of the cloud-fog system. These terminals, i.e. the low-level heuristics, are designed based on the global real-time information of the workflows and the cloud-fog system, so that the learned heuristics can solve the dynamic workflow scheduling.

Table I shows the terminal and function sets of CCGP. The terminals are the leaves of the tree while the functions cannot be located at the leaves of the trees. Note that the terminal set is different for each rule, but the function set is the same. 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.

No. 5 SD is the sub-deadline of a task, which is set based on the latest finish time, as shown in (17). We use reference execution time of a task α (e.g., terminal ET, denote as $ET(\alpha)$) and the transmission time in fog to estimate the latest finish time $\ell(\alpha)$ (18)). 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 [20], we set $\varepsilon = 0.95$. No. 6 UR is the upward rank of a task α, which is the length of the critical path from α to the exit task [44], as shown in (19).

$$SD(\alpha) = \begin{cases} \ell(\alpha), \text{ if } A^{suc}(\alpha) = \varnothing, \\ \varepsilon \times \ell(\alpha), \quad \text{otherwise.} \end{cases} \tag{17}$$

$$\ell(\alpha_1) = \begin{cases} \rho(g), & \text{if } A^{suc}(\alpha_1) = \varnothing, \\ \max\limits_{\alpha_2 \in A^{suc}(\alpha_1)} \Big\{ \ell(\alpha_2) - ET(\alpha_2) \\ - \frac{d(\alpha_1,\alpha_2)}{B_{LAN}} \Big\}, & \text{otherwise.} \end{cases} \tag{18}$$

$$UR(\alpha_1) = \begin{cases} ET(\alpha_1), & \text{if } A^{suc}(\alpha_1) = \varnothing, \\ ET(\alpha_1) + \max\limits_{\alpha_2 \in A^{suc}(\alpha_1)} \\ \Big\{ UR(\alpha_2) + \frac{d(\alpha_1,\alpha_2)}{B_{LAN}} \Big\}, & \text{otherwise.} \end{cases} \tag{19}$$

No. 19 CTR is the communication time of a task while it is executed on one resource, which is the time at which all predecessors' data is finally received. No. 20 ETAR is the minimum execution time of a task on a resource with the current remaining CPU and memory configuration. The current remaining CPU and memory of a resource (e.g. No. 21 CRCPU and No. 22 CRMEM) are calculated based on the sum of the minimum requirements of running tasks and waiting tasks on the resource. No. 23 FAAT is the factor of the task category, reflecting the complexity of the tasks assigned to a resource. If there is no task of the same category as the ready task on this resource, FAAT is equal to the number of other task categories. Otherwise, it is equal to the number of other task categories plus the total number of task categories (this paper considers 4 task categories). No. 24 AAT is the actual available time of a resource for a task 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. No. 25 CC is the communication cost of a task while it is executed on one resource, which is the total the cost for receiving data from all predecessors. Nos. 29-30 are container-related terminals. Since the CPU and memory of a container are configured separately, these terminals represent the corresponding values when configuring the container's CPU or memory.

V. Performance Evaluation and Results

A. Baseline Algorithms and Evaluation Metrics

The characteristics of the problem studied in this paper includes (1) dynamic workflow scheduling, (2) real-time resource allocation of containers (real-time Bin-packing problem), and (3) many-objective optimisation. To the best of our knowledge, no prior work has holistically addressed these three challenges simultaneously, leaving a gap in existing literature for directly comparable algorithms. Therefore, we use the following algorithms presented in related studies and the default scheduling strategy of container orchestration platforms as baseline algorithms. These algorithms are well-suited for addressing scheduling and resource allocation problems. Specifically, Spread, Binpacking, and Random were default scheduling strategies supported in Docker Swarm's initial release (v1.12), while only Spread remains in the latest version (v1.42). Spread achieves load balancing by evenly distributing tasks, Random provides a non-optimised baseline to evaluate overall scheduling efficiency, and Binpacking focuses on optimizing resource utilisation by minimizing idle space. *Spread strategy*: executes a container on the node having the least number of containers, which aims to ensure a good load balancing of containers between all nodes of the infrastructure. *Binpacking strategy*: executes a container on the most compacted node regarding resources in order to reduce the total number of used nodes. *Random strategy*: randomly selects a node from a set of nodes. *TOPSIS multi-criteria algorithm* [45]: select node based on three criteria by combining the Spread and Binpacking strategies: (i) the number of containers in each node; (ii) the number of available CPUs; and (iii) the amount of available memory. *Least Waste Fast First (LWFF)* [46]: select node using a multi-objective optimisation approach by considering simultaneous utilisation of memory and CPU separately for each node and cumulatively in the whole cluster.

Since these algorithms do not have the ability to be able to solve all the decisions of our problem, we combine these baseline algorithms with the Pareto solutions obtained from the CCGP training with different decisions to obtain the corresponding Pareto solutions, thus evaluating the performance of the baseline algorithms.

This paper uses two evaluation metrics to examine the performance of algorithms as follows:

1) *Hyper Volume (HV) [14], [47]:* HV evaluates the diversity and convergence of an evolutionary algorithm. It is obtained by calculating the volume of the enclosed area between a set of solutions generated by the algorithm and a reference point, which is usually selected as the maximum objective values. Specifically, we utilise Relative Percentage Deviation (RPD, Eq. (20)) to normalise three objectives values of solutions. The reference point $(1.0, 1.0, 1.0, 1.0)$ is used for calculating HV. Hence, the range of HV is $[0, 1]$ and a larger HV value is preferable, which indicates that the obtained set of nondominated solutions is closer to the Pareto front and has a desired distribution.

$$RPD^A = \frac{f^A - f_{\min}}{f_{\max} - f_{\min}}, \tag{20}$$

where f^A is the solution obtained by algorithm A, and $f_{\min}$ and $f_{\max}$ are the minimum and maximum values achieved among all algorithms, respectively.

2) *Inverted Generational Distance (IGD) [48], [49]:* The IGD evaluates the proximity between the optimal solutions obtained by the proposed methods and the true Pareto front, and is defined as:

$$IGD = \frac{1}{|Q|}\sum_{q\in Q}\min_{p\in P}\left\{\sqrt{\sum\nolimits_{i=1}^{m}(p_i - q_i)^2}\right\}, \tag{21}$$

where P is the set of Pareto optimal solutions obtained by the proposed methods, and Q is the set of true Pareto optimal solutions. m is the number of objectives. It is hard to know the true Pareto optimal solutions to the problem. To obtain approximate Pareto optimal solutions, we merge the results of all algorithms and then select the non-dominated solutions from this merged set as the approximate optimal solutions. A *smaller* IGD value is preferable, which indicates that the obtained set of nondominated solutions is closer to the approximated true Pareto front.

B. Simulation Environment Setup

To evaluate the performance of CCGP, we conduct a simulated cloud-fog environment based on the architecture proposed in Section II-A. Table II lists the configurations used in constructing the simulated cloud-fog environment. We select 5 representative resource types from low configuration to high configuration respectively, similar to [37]. These resource types are derived from AWS EC2,[1] and the associated power parameters are derived from Teads Engineering,[2] which provides the power consumption and carbon footprint estimator for AWS instances. Fog nodes have a limited number of nodes, whereas cloud servers do not limit the number of leases. To enhance resource elasticity in the simulation, container allocations of CPU or memory are configured in increments of 0.5 ECU or 0.5 GB. To account for practical deployment overheads, we incorporate a container initialisation latency of 50 ms prior to each task's execution [50]. Referring to [34], [35], the bandwidths of LAN and WAN are set to 0.5 GB/s and 2.0 GB/s, and the corresponding delays are set to 0.5 ms and 30 ms respectively. The unit price of data transmission is set to 0.2 \$/GB. The booting time of cloud servers is set to 55.9 s. AWS EC2 supports per-second billing[1], so we set the billing interval to 1s [20].

We consider two real-world data traces to extensive simulation experiments: one is three realistic scientific workflows from Pegasus project (including CyberShake, Montage, and Sipht); the other is the cluster data released by Alibaba Cloud in 2018[3].

[1]https://aws.amazon.com/ec2/

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

[3]https://download.pegasus.isi.edu/misc/SyntheticWorkflows.tar.gz;https://github.com/alibaba/clusterdata

TABLE II
CONFIGURATIONS OF FOG NODES AND CLOUD SEVERS[1,2]

Fog Nodes									Cloud Severs			
Type	(Numbers)	ECU	Memory (GB)	CPU-based idle power (Watt)	CPU-based full power (Watt)	Memory-based idle power (Watt)	Memory-based full power (Watt)	Base power (Watt)	Type	ECU	Memory (GB)	Cost ($/h)
m1.small	(3)	1	1.7	0.72	5.76	0.34	1.02	1.2	m4.xlarge	13	16	0.234
m3.medium	(3)	3	3.75	0.87	6.97	0.75	2.25	1.4	m4.2xlarge	26	32	0.468
m3.large	(4)	6.5	7.5	1.73	13.94	1.50	4.50	2.9	m4.4xlarge	53.5	64	0.936
m3.xlarge	(4)	13	15	3.47	27.87	3.00	9.00	5.8	m4.10xlarge	124.5	160	2.340
m3.2xlarge	(5)	26	30	6.94	55.74	6.00	18.00	11.5	m4.16xlarge	188	256	3.744

TABLE III
TRAINING AND TEST SCENARIOS

Parameter	Training	Test
Number of workflows based on Pegasus	15	10, 15, 20
Number of workflows based on Alibaba	50	30, 50, 100
Arrive rate	0.2, 0.8	

TABLE IV
THE PARAMETER SETTINGS OF CCGP [16], [20], [40]

Parameter	Value
Population size	3 subpopulations and 50 for each subpopulation
Number of generations	51
Method for initialising population	Ramped-half-and-half
Initial minimum/maximum depth	2 / 7
Elitism	5
Maximal depth	8
Crossover/mutation/reproduction rate	0.8/0.15/0.05
Terminal/non-terminal selection rate	10% / 90%

These established traces are widely adopted as standard benchmarks in fog computing research to ensure comparability and reproducibility [3], [16], [24], [51], [52], [53]. Furthermore, they collectively provide a diverse spectrum of workflow structures and scalability challenges that capture the essence of modern scheduling complexities. Although these public data provide the DAG structure and the execution time of each task, we need to construct data applicable to our problem based on the known information, such as the category of tasks and the CPU and memory of tasks. All the reconstructed data are available at https://github.com/zaixing-sun/CCGP_MDWS-CoCF. We classify the tasks of each workflow into four categories '00', '01', '10', and '11' in a ratio of 10%, 30%, 30% and 30%. The input and output 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 $\{0.8, 1.0, 1.5, 1.8\}$. The weight for each workflow is drawn from a uniform discrete distribution over the integer range [1,10]. Workflow arrival is modelled using the Poisson distribution with an arrival rate λ where the interarrival time is exponentially distributed with $1/\lambda$ [54], where $\lambda \in \{0.2, 0.8\}$.

The *name of each scenario* is set to $\langle \mathrm{wfNum}, \lambda \rangle$, where wfNum is the total number of workflows submitted and λ is the arrival rate. We train on four scenarios, $\langle 15, 0.2 \rangle$, $\langle 50, 0.2 \rangle$, $\langle 15, 0.8 \rangle$ and $\langle 50, 0.8 \rangle$, and then use the training results to test them according to the dataset and arrival rate, with a total of 12 test scenarios. The parameters of training and test scenarios are shown in Table III. We adopt literature-based values for parameters like the mutation and crossover rates [16], [20], [40]. To analyse the impact of subpopulation size, we conduct a sensitivity analysis (sizes 25, 50, and 100) which revealed a clear performance-cost trade-off. We find that increasing the subpopulation size from 50 to 100 offered only a marginal improvement in solution quality while nearly doubling the already significant computational time (e.g., from $\approx$2.5 to $\approx$4.2 hours on one scenario, and over 12 hours on more complex datasets). Therefore, a subpopulation size of 50 is selected for our experiments as it provides the best balance between high performance and computational feasibility, a finding that is consistent with the parameter settings in the related work of Xiao et al. [21]. The parameters of CCGP are shown in Table IV.

We implement the simulation and all the compared algorithms in Python 3.10 and C++ and deploy them on a computer with a 3.80 GHz Intel Core i7-10700 K processor and 15.3 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, which can be a good approximation of the true performance of the rule. It should be noted that the performance measures are obtained for each scenario by applying the evolved SPs thousands of times, since there are thousands of decisions needed to be made during a simulation instance of that scenario. For each scenario, there are a total of $30 \times 51 \times (50 \times 3) = 229500$ simulation instances during training, and each complete scheduling heuristic is tested on $30 \times 30 = 900$ simulation instances during testing. Using a large number of scenarios and simulation instances during the testing phase will help to confirm the quality and reusability of the evolved complete scheduling heuristic.

C. Results

Friedman's test with a significance level of 0.05 is applied to rank the algorithms based on their performance with 30 independent runs. If Friedman's test gives significance results, we further conduct the Wilcoxon rank-sum test with Bonferroni correction between the proposed algorithm and other algorithms with a significance level of 0.05 for the Nemenyi post-hoc pairwise comparisons [15], [29]. In the following results, "$\uparrow$", "$\downarrow$", and "$\approx$" indicate that the corresponding result is significantly better than, worse than, or similar to its counterpart. An algorithm will be compared with the algorithm(s) before it one

TABLE V
THE MEAN AND STANDARD DEVIATION OF THE HV VALUES OVER THE 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS

$\langle$wfNum, $\lambda\rangle$	Random	Binpacking	Spread	TOPSIS	LWFF	CCGP
$\langle 10, 0.2\rangle$	0.020±0.011	0.100±0.032(↑)	0.138±0.035(↑)(↑)	0.001±0.001(↓)(↓)(↓)	0.350±0.093(↑)(↑)(↑)(↑)	0.811±0.061(↑)(↑)(↑)(↑)(↑)
$\langle 15, 0.2\rangle$	0.022±0.014	0.120±0.035(↑)	0.115±0.033(↑)(≈)	0.002±0.002(↓)(↓)(↓)	0.441±0.099(↑)(↑)(↑)(↑)	0.788±0.066(↑)(↑)(↑)(↑)(↑)
$\langle 20, 0.2\rangle$	0.024±0.016	0.111±0.036(↑)	0.101±0.028(↑)(≈)	0.002±0.002(↓)(↓)(↓)	0.473±0.091(↑)(↑)(↑)(↑)	0.769±0.066(↑)(↑)(↑)(↑)(↑)
$\langle 30, 0.2\rangle$	0.036±0.021	0.025±0.022(↓)	0.033±0.019(≈)(↑)	0.068±0.034(↑)(↑)(↑)	0.492±0.103(↑)(↑)(↑)(↑)	0.762±0.077(↑)(↑)(↑)(↑)(↑)
$\langle 50, 0.2\rangle$	0.047±0.019	0.019±0.015(↓)	0.035±0.026(↓)(↑)	0.082±0.030(↑)(↑)(↑)	0.482±0.108(↑)(↑)(↑)(↑)	0.812±0.065(↑)(↑)(↑)(↑)(↑)
$\langle 100, 0.2\rangle$	0.047±0.021	0.022±0.018(↓)	0.032±0.021(↓)(↑)	0.050±0.033(≈)(↑)(↑)	0.531±0.086(↑)(↑)(↑)(↑)	0.789±0.074(↑)(↑)(↑)(↑)(↑)
$\langle 10, 0.8\rangle$	0.016±0.007	0.066±0.021(↑)	0.148±0.046(↑)(↑)	0.001±0.002(↓)(↓)(↓)	0.350±0.084(↑)(↑)(↑)(↑)	0.823±0.080(↑)(↑)(↑)(↑)(↑)
$\langle 15, 0.8\rangle$	0.020±0.008	0.086±0.020(↑)	0.140±0.042(↑)(↑)	0.002±0.002(↓)(↓)(↓)	0.409±0.081(↑)(↑)(↑)(↑)	0.789±0.098(↑)(↑)(↑)(↑)(↑)
$\langle 20, 0.8\rangle$	0.021±0.008	0.087±0.018(↑)	0.130±0.038(↑)(↑)	0.002±0.003(↓)(↓)(↓)	0.429±0.076(↑)(↑)(↑)(↑)	0.764±0.095(↑)(↑)(↑)(↑)(↑)
$\langle 30, 0.8\rangle$	0.021±0.009	0.017±0.008(↓)	0.019±0.015(≈)(≈)	0.051±0.024(↑)(↑)(↑)	0.476±0.076(↑)(↑)(↑)(↑)	0.818±0.077(↑)(↑)(↑)(↑)(↑)
$\langle 50, 0.8\rangle$	0.018±0.010	0.023±0.013(↑)	0.020±0.015(≈)(↓)	0.067±0.018(↑)(↑)(↑)	0.480±0.099(↑)(↑)(↑)(↑)	0.843±0.078(↑)(↑)(↑)(↑)(↑)
$\langle 100, 0.8\rangle$	0.018±0.013	0.040±0.021(↑)	0.022±0.016(≈)(↓)	0.026±0.021(↑)(↓)(≈)	0.538±0.095(↑)(↑)(↑)(↑)	0.805±0.101(↑)(↑)(↑)(↑)(↑)
W/D/L	0/0/12	0/0/12	0/0/12	0/0/12	0/0/12	N/A
AverageRank	4.786	4.375	4.108	4.714	2.017	1.000

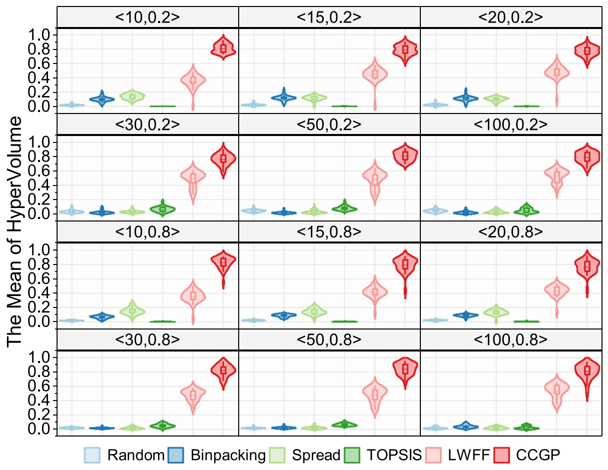


Fig. 5. Violin plots of the HV over 30 independent runs on unseen instances.

by one. "Win, Draw, Lose" means the number of scenarios that a compared algorithm is statistically better, similar, or worse than CCGP. "Average Rank" shows the average ranking of the algorithm on all the examined scenarios.

1) Comparison of the HyperVolume: Table V shows the mean and standard deviation of the HV values over the 30 independent runs of the compared algorithms. Overall, CCGP is the best algorithm based on the average ranking according to the Friedman test. CCGP achieves the highest HV values across all scenarios, indicating its ability to generate diverse, high-quality solutions close to the Pareto front. For example, with 10 workflows and an arrival rate of 0.2 (e.g. scenario $\langle 10, 0.2\rangle$), CCGP's mean HV is 0.811, significantly higher than the best baseline, LWFF, at 0.350. With 100 workflows (e.g. scenario $\langle 100, 0.2\rangle$), CCGP has a mean HV of 0.789, compared to the best baseline at 0.531. Even with an arrival rate of 0.8 (e.g. scenario $\langle 100, 0.8\rangle$), CCGP's mean HV is 0.805, well above the closest baseline at 0.538. The Random strategy has the lowest overall performance, with an average rank of 4.786, due to its lack of optimisation criteria. Binpacking has mixed results, focusing on minimising active nodes without load balancing. The Spread strategy achieves moderate performance by distributing containers evenly, but failing to optimise for specific resource requirements. The TOPSIS performs the worst, with an average rank of 4.714, often scoring low due to its lack of adaptive capabilities. LWFF is the best baseline algorithm, with an average rank of 2.017, using a multi-objective optimisation approach for efficient resource allocation. However, it still fails to match the performance of CCGP, which consistently outperforms other methods in terms of robustness and effectiveness.

Fig. 5 shows the violin plots of the HV values over the 30 independent runs of the compared algorithms on unseen instances. CCGP shows a higher concentration of HV values near the maximum, indicating consistently high performance. In contrast, the other algorithms show wider distributions with lower median HV values, reflecting their inconsistent performance and inability to consistently find high-quality solutions. For scenarios with 10 workflows and arrival rates of 0.2 and 0.8, the violin plots show CCGP with tightly clustered high HV values, while other algorithms, such as Random and TOPSIS show broader distributions with lower values. As the number of workflows increases, the superiority of CCGP becomes more pronounced, with its HV values remaining consistently high, unlike the fluctuating performance of other algorithms.

In summary, CCGP's dynamic evolution of scheduling heuristics based on real-time system states allows it to generate high-quality solutions well-distributed along the Pareto front. This adaptability significantly outperforms traditional heuristic and TOPSIS methods, with CCGP consistently ranking at the top in all scenarios. The average rank of 1.003 and CCGP winning in all 12 scenarios confirm its robustness and superiority for dynamic workflow scheduling in container-based cloud-fog environments.

2) Comparison of the Inverted Generational Distance: Table VI shows the mean and standard deviation of the IGD values over the 30 independent runs of the compared algorithms. Overall, CCGP is again the best algorithm based on the average ranking according to the Friedman test. CCGP consistently achieves the lowest IGD values across all scenarios, indicating its ability to generate solutions closest to the true Pareto front. The TOPSIS algorithm, which combines multiple criteria, performs the worst with an average rank of 5.168, and often scores high IGD values due to its inability to adapt to

TABLE VI
THE MEAN AND STANDARD DEVIATION OF THE IGD VALUES OVER THE 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS

$\langle$wfNum, $\lambda\rangle$	Random	Binpacking	Spread	TOPSIS	LWFF	CCGP
$\langle 10, 0.2\rangle$	0.607±0.084	0.473±0.088(↑)	0.642±0.071(↓)(↓)	1.146±0.098(↓)(↓)(↓)	0.743±0.140(↓)(↓)(↓)(↑)	0.285±0.106(↑)(↑)(↑)(↑)(↑)
$\langle 15, 0.2\rangle$	0.602±0.095	0.470±0.093(↑)	0.698±0.091(↓)(↓)	1.123±0.103(↓)(↓)(↓)	0.685±0.129(↓)(↓)(≈)(↑)	0.279±0.078(↑)(↑)(↑)(↑)(↑)
$\langle 20, 0.2\rangle$	0.602±0.087	0.488±0.082(↑)	0.710±0.080(↓)(↓)	1.106±0.106(↓)(↓)(↓)	0.651±0.114(≈)(↓)(↑)(↑)	0.274±0.073(↑)(↑)(↑)(↑)(↑)
$\langle 30, 0.2\rangle$	0.664±0.074	0.665±0.077(≈)	0.714±0.100(↓)(↓)	0.699±0.095(↓)(≈)(≈)	0.728±0.107(↓)(↓)(≈)(≈)	0.312±0.091(↑)(↑)(↑)(↑)(↑)
$\langle 50, 0.2\rangle$	0.630±0.089	0.653±0.097(↓)	0.811±0.097(↓)(↓)	0.685±0.089(↓)(≈)(↑)	0.695±0.134(≈)(≈)(↑)(≈)	0.330±0.123(↑)(↑)(↑)(↑)(↑)
$\langle 100, 0.2\rangle$	0.651±0.087	0.692±0.104(↓)	0.823±0.107(↓)(↓)	0.778±0.105(↓)(↓)(↑)	0.673±0.138(≈)(≈)(↑)(↑)	0.318±0.104(↑)(↑)(↑)(↑)(↑)
$\langle 10, 0.8\rangle$	0.600±0.087	0.477±0.102(↑)	0.620±0.074(≈)(↓)	1.165±0.090(↓)(↓)(↓)	0.718±0.123(↓)(↓)(↓)(↑)	0.286±0.124(↑)(↑)(↑)(↑)(↑)
$\langle 15, 0.8\rangle$	0.604±0.101	0.496±0.109(↑)	0.647±0.080(≈)(↓)	1.144±0.097(↓)(↓)(↓)	0.669±0.128(≈)(↓)(≈)(↑)	0.291±0.117(↑)(↑)(↑)(↑)(↑)
$\langle 20, 0.8\rangle$	0.615±0.082	0.523±0.103(↑)	0.664±0.090(↓)(↓)	1.126±0.098(↓)(↓)(↓)	0.638±0.104(≈)(↓)(≈)(↑)	0.280±0.094(↑)(↑)(↑)(↑)(↑)
$\langle 30, 0.8\rangle$	0.672±0.095	0.641±0.113(↑)	0.869±0.139(↓)(↓)	0.725±0.093(↓)(↓)(↑)	0.667±0.100(≈)(≈)(↑)(↑)	0.321±0.095(↑)(↑)(↑)(↑)(↑)
$\langle 50, 0.8\rangle$	0.652±0.104	0.615±0.121(↑)	0.917±0.122(↓)(↓)	0.692±0.083(↓)(↓)(↑)	0.645±0.104(≈)(≈)(↑)(↑)	0.309±0.101(↑)(↑)(↑)(↑)(↑)
$\langle 100, 0.8\rangle$	0.686±0.097	0.623±0.120(↑)	0.915±0.141(↓)(↓)	0.867±0.104(↓)(↓)(↑)	0.644±0.106(≈)(≈)(↑)(↑)	0.315±0.110(↑)(↑)(↑)(↑)(↑)
W/D/L	0/0/12	0/0/12	0/0/12	0/0/12	0/0/12	N/A
AverageRank	3.514	2.707	4.732	5.168	3.768	1.111

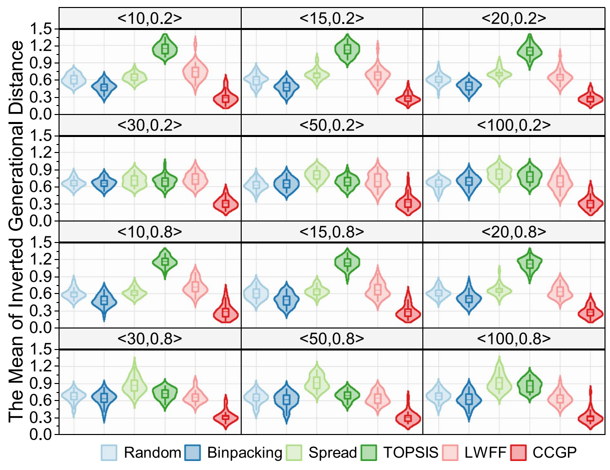


Fig. 6. Violin plots of the IGD over 30 independent runs on unseen instances.

changing workloads. The Spread strategy also performs poorly by distributing containers evenly but fails to optimise specific resource requirements, resulting in wider IGD distributions. The Random strategy shows moderate performance due to its lack of optimisation criteria, resulting in high IGD values and poor proximity to the Pareto front. Binpacking, which focuses on minimising active nodes, performs better on IGD than HV. This discrepancy may be due to Binpacking's focus on resource efficiency, which limits its exploration of a wide range of optimal solutions, thereby compromising its ability to achieve diverse and high-quality solutions. LWFF has mixed results, suggesting it can generate solutions close to the true Pareto front but struggles with solution diversity.

Fig. 6 shows the violin plots of the IGD over 30 independent runs on unseen instances. The results further confirm the superior performance of CCGP, with the smallest IGD values and the narrowest distributions, indicating its ability to generate high-quality solutions close to the true Pareto front. As the number of workflows increases, CCGP's superiority becomes more pronounced, consistently maintaining low IGD values.

In summary, CCGP's dynamic evolution of scheduling heuristics based on real-time system states allows it to generate solutions that are closest to the true Pareto front. This adaptability significantly outperforms traditional heuristic and TOPSIS optimisation methods. The average ranks and consistent wins in all scenarios for both HV and IGD metrics validate its robustness and superiority for dynamic workflow scheduling in container-based cloud-fog environments.

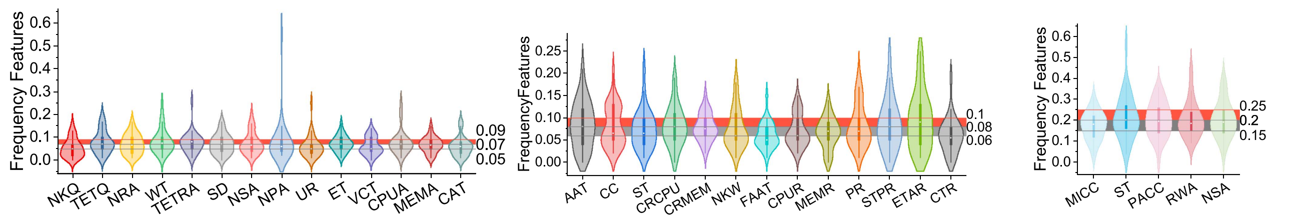
D. Insight of Evolved Scheduling Heuristics

To gain further understanding of the behaviour of the scheduling heuristic evolved by the proposed method, evolved scheduling heuristics are selected to be analysed.

1) Frequency Features: Fig. 7 shows the violin plots of the average terminals frequency of three rules learned by CCGP over the 30 independent runs in scenario $\langle 50, 0.8\rangle$. By default, all features associated with different rules have the same probability of being selected initially. For the *task selection rule* (Fig. 7(a)), the terminal frequencies are highly skewed. A few key features (e.g., WT, TETRA, SD, ET) are used with high frequency while the majority show minimal usage, which demonstrates the evolution of a highly specialized and feature-selective policy. In contrast, for both the *resource selection rule* (Fig. 7(b)) and *container deployment rule* (Fig. 7(c)), the frequency distribution of almost all features is widest within a range of 25% around the initial probability, and the medians are also distributed within this range. This suggests that the learned rules not only use the available information effectively, but also exploit the interactions and combinations of different features, avoiding over-reliance on any single feature.

In single-objective optimisation problems, such as those studied by [20], [55], algorithm evolution can filter out features with different frequencies, with higher frequencies favouring single-objective optimisation. However, in the multi-objective optimisation problem considered in this paper, algorithm evolution focuses more on balancing multiple objectives. Therefore, the feature frequencies do not deviate significantly from the initial probability, reflecting the need to maintain a diverse set of solutions that satisfy different trade-offs between objectives. This balanced use of features, together with effective combinations and interactions, ensures that the evolved heuristics are robust and adaptable to different scenarios and requirements.

2) Learned Pareto Fronts and Scheduling Heuristics: Fig. 8 shows the parallel coordinates plot of non-dominated solutions obtained from different algorithms for many-objective in scenario $\langle 50, 0.8\rangle$. This plot provides a visual comparison of how each algorithm performs across four objectives: Total Cost, Total



(a) The frequency of terminals in *task selection rule.* (b) The frequency of terminals in *resource selection rule.* (c) The frequency of terminals in *container deployment rule.*

Fig. 7. Violin plots of the average terminals frequency of three rules learned by CCGP over the 30 independent runs in scenario $\langle 50, 0.8 \rangle$.

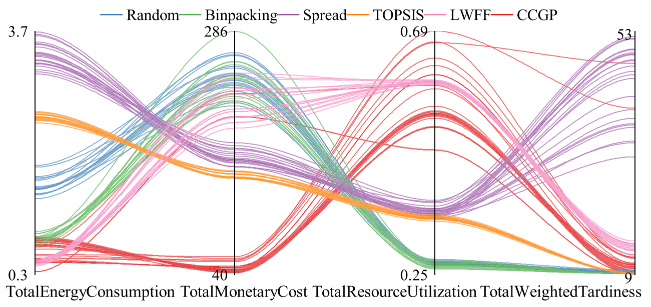


Fig. 8. The parallel coordinates plot of non-dominated solutions obtained from different algorithms for many-objective in scenario $\langle 50, 0.8 \rangle$.

Energy Consumption, Resource Utilisation, and Total Weighted Tardiness. CCGP consistently achieves superior performance across all objectives, with lines favourably positioned on each axis. It shows the lowest total cost and energy consumption, high resource utilisation, and significantly reduced total weighted tardiness. Baseline algorithms show varied performance. Random and Binpacking strategies perform poorly for many objectives. Spread performs moderately but lacks comprehensive optimisation. TOPSIS and LWFF improve slightly but fall short of CCGP's balanced optimisation. TOPSIS slightly outperforms LWFF on cost, energy, and tardiness, but at the expense of slightly lower resource utilisation. The poor resource utilisation by Binpacking and Spread strategies exposes the limitations of greedy approaches in multi-objective optimisation. Binpacking minimises active nodes but leads to uneven load distribution, while Spread balances load but fails to optimise resource allocation.

VI. Conclusion

This paper considers a more fine-grained many-objective dynamic workflow scheduling in container-based cloud-fog computing, which supports both vertical and horizontal scaling of containers and executes tasks of various categories in adjustable configurations containers according to task requirements. To the best of our knowledge, this is the first work on dynamic workflow scheduling in container-based cloud-fog computing with vertical and horizontal scaling of containers. To address this novel and challenging problem, we develop a dynamic scheduling simulator to simulates the workflow 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 approach can evolve the three scheduling heuristics (i.e., task selection rule, resource selection rule and container deployment rule) simultaneously to meet these decision points. CCGP is examined based on two real-world data traces. The simulation results and comparisons show that it significantly outperforms all baseline algorithms on all tested scenarios in terms of HV and IGD, which suggests that CCGP is capable of evolving near-opimal, diverse, and uniformly distributed scheduling heuristics for generating the corresponding Pareto fronts. We also find that the scheduling heuristics evolved via CCGP do not rely on a particular feature, but rather focus on the effective combination and interaction of features to ensure that the CCGP is robust and can be adapted to different dynamic scenarios and requirements, which cannot be achieved by simple combinations of the existing scheduling rules to tackle many objectives.

The proposed framework is highly extensible, featuring a problem model that supports diverse task types and elastic container scaling, and a modular simulator that allows for the seamless replacement of scheduling heuristics at key decision points. While the proposed CCGP algorithm demonstrates superior performance, it still has three main limitations to be addressed in future work.

Firstly, our evaluation relies on established benchmark datasets (Pegasus and Alibaba traces) due to the current lack of public workflow traces from native fog environments. Although these datasets ensure comparability and test scalability, a key direction for future work is to evaluate the CCGP algorithm's performance on authentic fog-native workloads, particularly under dynamic workload distributions. *Secondly*, our simulation model abstracts certain real-world orchestration complexities. While we incorporated a constant container initialisation overhead, future work should model variable startup times and explore the integration of our scheduler with advanced mitigation strategies, such as container reuse and pre-warmed pools, to better reflect production environments. *Finally*, our system model assumes a fully connected network topology, a common simplification that allows us to focus on the core scheduling logic. Future research should assess the robustness of the CCGP algorithm under more realistic network conditions, such as constrained or hierarchical topologies, where factors like network congestion play a critical role. *In addition* to addressing these limitations, we also plan to explore enhancing the interpretability of the evolved heuristics within the CCGP approach to improve the transparency and practical applicability of the decision-making process.

References

[1] M. Goudarzi, M. A. Rodriguez, M. Sarvi, and R. Buyya, "μ-DDRL: A QoS-Aware distributed deep reinforcement learning technique for service offloading in fog computing environments," *IEEE Trans. Serv. Comput.*, vol. 17, no. 1, pp. 47–59, Jan./Feb. 2024.

[2] S. O. Ogundoyin and I. A. Kamil, "Optimization techniques and applications in fog computing: An exhaustive survey," *Swarm Evol. Comput.*, vol. 66, 2021, Art. no. 100937.

[3] S. Karami, S. Azizi, and F. Ahmadizar, "A BI-objective workflow scheduling in virtualized fog-cloud computing using NSGA-II with semi-greedy initialization," *Appl. Soft Comput.*, vol. 151, 2024, Art. no. 111142.

[4] A. Ghammam, T. Ferreira, W. Aljedaani, M. Kessentini, and A. Husain, "Dynamic software containers workload balancing via many-objective search," *IEEE Trans. Serv. Comput.*, vol. 16, no. 4, pp. 2575–2591, Jul./Aug. 2023.

[5] Z. Li, H. Yu, G. Fan, and J. Zhang, "Cost-efficient fault-tolerant workflow scheduling for deadline-constrained microservice-based applications in clouds," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 3, pp. 3220–3232, Sep. 2023.

[6] J. Luo et al., "Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT," *Future Gener. Comput. Syst.*, vol. 97, pp. 50–60, 2019.

[7] G. Sheganaku, S. Schulte, P. Waibel, and I. Weber, "Cost-efficient auto-scaling of container-based elastic processes," *Future Gener. Comput. Syst.*, vol. 138, pp. 296–312, 2023.

[8] "Resize CPU and memory resources assigned to containers," Kubernetes Documentation, 2024. Accessed: Jul. 24, 2024. [Online]. Available: https://kubernetes.io/docs/tasks/configure-pod-container/resize-container-resources/

[9] "Runtime options with memory, cpus, and GPUs," Docker Docs, 2024. Accessed: Jul. 24, 2024. [Online]. Available: https://docs.docker.com/config/containers/resource_constraints/

[10] M. A. Rodriguez and R. Buyya, "Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms," *Future Gener. Comput. Syst.*, vol. 79, pp. 739–750, 2018.

[11] 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, Jul. 2022.

[12] Z. Sun et al., "Evolving scheduling heuristics for energy-efficient dynamic workflow scheduling in cloud via genetic programming hyper-heuristics," in *Proc. Int. Conf. Intell. Comput.*, 2024, pp. 169–182.

[13] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Apr.–Jun. 2014.

[14] Z.-G. Chen et al., "Multiobjective cloud workflow scheduling: A multiple populations ant colony system approach," *IEEE Trans. Cybern.*, vol. 49, no. 8, pp. 2912–2926, Aug. 2019.

[15] Z.-J. Wang et al., "Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2715–2729, Jun. 2020.

[16] M. Xu et al., "Genetic programming for dynamic workflow scheduling in fog computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 4, pp. 2657–2671, Jul./Aug. 2023.

[17] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach," *J. Netw. Comput. Appl.*, vol. 201, 2022, Art. no. 103333.

[18] B. Tan, H. Ma, Y. Mei, and M. Zhang, "A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in container-based clouds," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1500–1514, Jul.–Sep. 2022.

[19] 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, May/Jun. 2023.

[20] 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, Sep./Oct. 2024.

[21] Q.-Z. Xiao, J. Zhong, L. Feng, L. Luo, and J. Lv, "A cooperative coevolution hyper-heuristic framework for workflow scheduling problem," *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 150–163, Jan./Feb. 2022.

[22] Q.-H. Zhu, H. Tang, J.-J. Huang, and Y. Hou, "Task scheduling for multi-cloud computing subject to security and reliability constraints," *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 4, pp. 848–865, Apr. 2021.

[23] X. Tang, "Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2909–2919, Oct.–Dec. 2022.

[24] A. Taghinezhad-Niar and J. Taheri, "Security, reliability, cost, and energy-aware scheduling of real-time workflows in compute-continuum environments," *IEEE Trans. Cloud Comput.*, vol. 12, no. 3, pp. 954–965, Jul.–Sept. 2024.

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

[26] 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, Feb. 2016.

[27] Y. Yang, G. Chen, H. Ma, S. Hartmann, and M. Zhang, "Dual-tree genetic programming with adaptive mutation for dynamic workflow scheduling in cloud computing," *IEEE Trans. Evol. Comput.*, vol. 29, no. 5, pp. 1692–1706, Oct. 2025, doi: 10.1109/TEVC.2024.3392968.

[28] Y. Mei, S. Nguyen, B. Xue, and M. Zhang, "An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 1, no. 5, pp. 339–353, Oct. 2017.

[29] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask genetic programming-based generative hyperheuristics: A case study in dynamic scheduling," *IEEE Trans. Cybern.*, vol. 52, no. 10, pp. 10515–10528, Oct. 2022.

[30] M. A. Ardeh, Y. Mei, and M. Zhang, "Genetic programming with knowledge transfer and guided search for uncertain capacitated arc routing problem," *IEEE Trans. Evol. Comput.*, vol. 26, no. 4, pp. 765–779, Aug. 2022.

[31] S. Wang, Y. Mei, and M. Zhang, "Explaining genetic programming-evolved routing policies for uncertain capacitated arc routing problems," *IEEE Trans. Evol. Comput.*, vol. 28, no. 4, pp. 918–932, Aug. 2024.

[32] Y. J. Yao, Q. H. Liu, X. Y. Li, and L. Gao, "A novel MILP model for job shop scheduling problem with mobile robots," *Robot. Comput. Integr. Manuf.*, vol. 81, 2023, Art. no. 102506.

[33] L. M. Antonio and C. A. C. Coello, "Coevolutionary multiobjective evolutionary algorithms: Survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 22, no. 6, pp. 851–865, Dec. 2018.

[34] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent IoT applications in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1298–1311, Apr. 2021.

[35] S. Pallewatta, V. Kostakos, and R. Buyya, "QoS-aware placement of microservices-based IoT applications in fog computing environments," *Future Gener. Comput. Syst.*, vol. 131, pp. 121–136, 2022.

[36] D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan, "Learn-as-you-go with megh: Efficient live migration of virtual machines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1786–1801, Aug. 2019.

[37] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings, "COSCO: Container orchestration using co-simulation and gradient based optimization for fog computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 1, pp. 101–116, Jan. 2022.

[38] 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. Mobile Comput.*, vol. 22, no. 4, pp. 2116–2130, Apr. 2023.

[39] F. Zhang, S. Nguyen, Y. Mei, and M. Zhang, *Genetic Programming for Production Scheduling*. Singapore: Springer, 2021.

[40] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Trans. Evol. Comput.*, vol. 18, no. 2, pp. 193–208, Apr. 2014.

[41] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[42] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Aug. 2014.

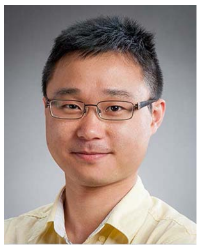
[43] H. Jain and K. Deb, "An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 602–622, Aug. 2014.

[44] 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, Mar. 2002.

[45] T. Menouer and P. Darmon, “New Scheduling Strategy Based on Multi-Criteria Decision Algorithm,” in *Proc. 27th Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, 2019, pp. 101–107.
[46] H. M. Fard, R. Prodan, and F. Wolf, “Dynamic multi-objective scheduling of microservices in the cloud,” in *Proc. IEEE/ACM 13th Int. Conf. Utility Cloud Comput.*, 2020, pp. 386–393.
[47] L. Pan, X. Liu, Z. Jia, J. Xu, and X. Li, “A multi-objective clustering evolutionary algorithm for multi-workflow computation offloading in mobile edge computing,” *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1334–1351, Apr.–Jun. 2023.
[48] X. Cai, S. Geng, D. Wu, J. Cai, and J. Chen, “A multicloud-model-based many-objective intelligent algorithm for efficient task scheduling in Internet of Things,” *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9645–9653, Jun. 2021.
[49] C. Song, H. Zheng, G. Han, P. Zeng, and L. Liu, “Cloud edge collaborative service composition optimization for intelligent manufacturing,” *IEEE Trans. Ind. Informat.*, vol. 19, no. 5, pp. 6849–6858, May 2023.
[50] R. Ranjan, I. S. Thakur, G. S. Aujla, N. Kumar, and A. Y. Zomaya, “Energy-efficient workflow scheduling using container-based virtualization in software-defined data centers,” *IEEE Trans. Ind. Informat.*, vol. 16, no. 12, pp. 7646–7657, Dec. 2020.
[51] G. Singh and A. K. Chaturvedi, “Hybrid modified particle swarm optimization with genetic algorithm (GA) based workflow scheduling in cloud-fog environment for multi-objective optimization,” *Cluster Comput.*, vol. 27, no. 2, pp. 1947–1964, 2024.
[52] S. Nazemi and R. Khorsand, “C-KHCS: Multi-objective workflow scheduling using chaotic krill herd optimization and improved cuckoo search in fog computing,” *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2095–2108, Sep./Oct. 2024.
[53] E. Oustad et al., “DIST: Distributed learning-based energy-efficient and reliable task scheduling and resource allocation in fog computing,” *IEEE Trans. Serv. Comput.*, vol. 18, no. 3, pp. 1336–1351, May/Jun. 2025.
[54] S. Sahoo, B. Sahoo, and A. K. Turuk, “A learning automata-based scheduling for deadline sensitive task in the cloud,” *IEEE Trans. Serv. Comput.*, vol. 14, no. 6, pp. 1662–1674, Nov./Dec. 2021.
[55] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, “Genetic programming with adaptive search based on the frequency of features for dynamic flexible job shop scheduling,” in *Proc. 21st Eur. Conf. Evol. Comput. Combinatorial Optim.*, 2020, pp. 214–230.

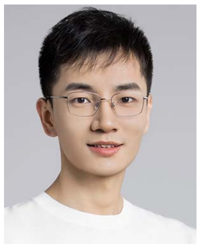
Zaixing Sun received the PhD degree in computer science and technology from the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He is currently a postdoctoral research fellow with the Department of New Networks, Pengcheng Laboratory, Shenzhen. His research interests include cloud computing, intelligent optimisation and scheduling.

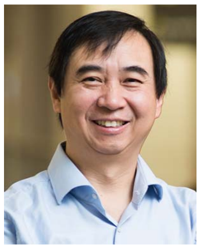
Fangfang Zhang (Member, IEEE) received the BSc and MSc degrees from Shenzhen University, Shenzhen, China, in 2014 and 2017, respectively, and the PhD degree in computer science from the Victoria University of Wellington, New Zealand, in 2021. She is currently a lecturer of artificial intelligence with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. Her research interests include evolutionary computation, hyper-heuristic learning/optimisation, job shop scheduling, surrogate, and multitask learning.


Yi Mei (Senior Member, IEEE) received the BSc and PhD degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively. He is currently an associate professor with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary computation for combinatorial optimisation, genetic programming, and hyper-heuristic. He is an associate editor for *IEEE Transactions on Evolutionary Computation*, and a guest editor of the *Genetic Programming Evolvable Machine* journal. He is a fellow of Engineering New Zealand.

Hejiao Huang received the PhD degree in computer science from the City University of Hong Kong, in 2004. She was an invited professor with INRIA, France. She is currently a professor with the Harbin Institute of Technology, Shenzhen, China. Her research interests include network security, cloud computing security, trustworthy computing, Big Data security, formal methods for system design, and wireless networks.

Chonglin Gu received the PhD degree in computer science and technology from the Harbin Institute of Technology, Shenzhen, in 2018. He was a postdoctoral fellow with the Chinese University of Hong Kong, Shenzhen, China. He is currently an assistant professor with the School of Computer Science and Technology, Harbin Institute of Technology. His research interests include cloud computing, especially algorithm design and system implementation.


Bin Wang received the PhD degree from the South China University of Technology, in 2021. He is currently an assistant research fellow with the Department of New Networks, Pengcheng Laboratory, Shenzhen, China. His research interests include energy-efficient computing architectures and scheduling optimization in cloud/edge.


Mengjie Zhang (Fellow, IEEE) received the BE and ME degrees from Artificial Intelligence Research Center, Agricultural University of Hebei, Baoding, China, in 1989 and 1992, respectively, and the PhD degree in computer science from RMIT University, Melbourne, VIC, Australia, in 2000. He is currently a professor of computer science, Head of the Evolutionary Computation Research Group. His research interests include evolutionary computation, genetic programming, multi-objective optimisation, and job shop scheduling. Prof. Zhang is a fellow of Royal Society of New Zealand, a fellow of Engineering New Zealand, and an IEEE CIS distinguished lecturer.