electronics

MDPI

Article

Towards Efficient Job Scheduling for Cumulative Data Processing in Multi-Cloud Environments

Yi Liang *, Guimei Xu, Haotian Shen, Nianyi Ruan and Yinzhou Wang

College of Computer Science, Beijing University of Technology, Beijing 100124, China; guimei@emails.bjut.edu.cn (G.X.); shenht@emails.bjut.edu.cn (H.S.); s202274131@emails.bjut.edu.cn (N.R.); wyz555@emails.bjut.edu.cn (Y.W.)

* Correspondence: yliang@bjut.edu.cn

Abstract: The rapid expansion of multi-cloud environments enables the fulfillment of the dynamic and diverse resource requirements of cloud applications. Cumulative data processing (CDP) applications, which handle incrementally generated data in stages like preprocessing and aggregate analysis, particularly benefit from these environments. However, existing cloud scheduling solutions struggle to handle the dynamic accumulation of processed data and the long-term data operation dependencies in CDP applications. Aiming at this issue, we propose a novel job execution model, CDP-EM, and a tailored job scheduling strategy, CDP-JS, to optimize the scheduling of CDP applications in multi-cloud environments. The CDP-EM model enables dynamic job generation and dependency-aware execution for CDP applications, while the CDP-JS strategy formulates the job scheduling problem as a Markov Decision Process (MDP), utilizing deep reinforcement learning with Proximal Policy Optimization (PPO) to optimize scheduling decisions. The simulation results show that integrating CDP-EM and CDP-JS reduces the SLA violation rate and resource cost of CDP applications by an average of 34.8% and 23.4%, respectively. Real-world evaluations show average reductions of 27.2% and 31.3%, respectively.

Keywords: multi-cloud; cumulative data processing; job scheduling; deep reinforcement learning

Academic Editor: Stanley C. Ahalt

Received: 17 February 2025
Revised: 18 March 2025
Accepted: 25 March 2025
Published: 27 March 2025

Citation: Liang, Y.; Xu, G.; Shen, H.; Ruan, N.; Wang, Y. Towards Efficient Job Scheduling for Cumulative Data Processing in Multi-Cloud Environments. *Electronics* **2025**, *14*, 1332. https://doi.org/10.3390/electronics14071332

1. Introduction

The widespread adoption of cloud computing has fueled the emergence of numerous cloud service providers, delivering on-demand access to computing resources and services through flexible, pay-as-you-go models [1]. However, single-cloud environments are increasingly unable to address the diverse and dynamic resource requirements of cloud applications; for example, in deep learning tasks, such as image classification, data preprocessing demands significant storage and CPU resources, which grow with data volume. A single-cloud environment may struggle to scale these resources, and during model training, GPU demand spikes. A single-cloud provider might not offer enough GPUs, limiting performance [2]. As a result, users are turning to more adaptable and cost-efficient multi-cloud solutions. By combining resources from multiple providers, multi-cloud environments allow users to optimize costs, enhance flexibility, and better accommodate diverse application needs. This shift is driving the evolution from single-cloud to multi-cloud architectures [3].

Cumulative data processing (CDP) applications manage data that are generated incrementally over time and typically operate in two key stages: preprocessing and aggregate analysis. In the preprocessing stage, data are independently transformed and prepared for subsequent use. In the aggregate analysis stage, the preprocessed data are combined to

perform complex analytical tasks, which must be completed within a specified deadline. For instance, in e-commerce, user purchase behavior data are collected throughout the day, processed using ETL pipelines, and then aggregated overnight to generate user behavior profiles via deep learning models, as shown in Figure 1. These profiles, available by early morning, help shape sales strategies and enhance the user's experience. Similar workflows can be observed in social media sentiment analysis, IoT device monitoring, etc. Given the varying resource demands across stages and their long lifecycles, CDP applications can effectively utilize low-cost and heterogeneous resources across multiple clouds, making them particularly well-suited for multi-cloud environments.

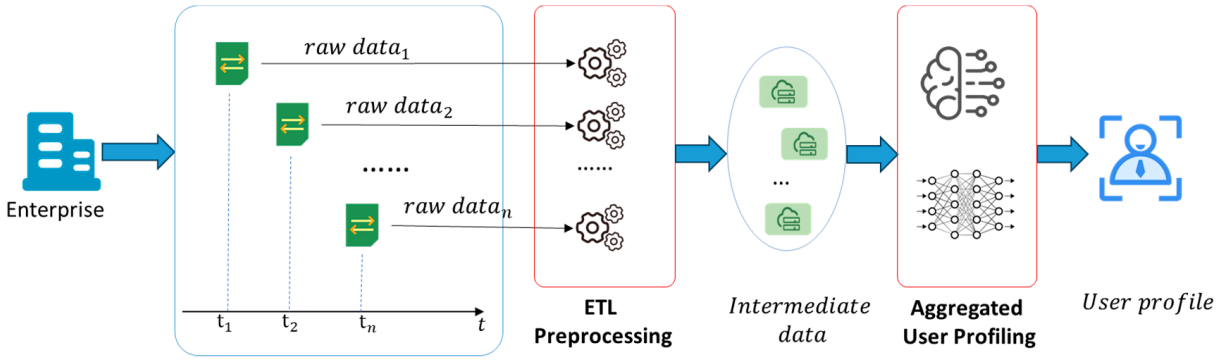


Figure 1. Example of CDP application: User purchase behavior analysis in E-commerce.

In multi-cloud environments, applications are typically composed of one or more jobs, which serve as the basic units of scheduling and execution. The job execution model and job scheduling strategy serve as the two fundamental components of a comprehensive cloud job scheduling solution. The job execution model refers to a framework that determines how cloud jobs are structured, scheduled, and executed. Existing job execution models in cloud computing can be broadly classified into basic models and workflow models. The basic model treats cloud jobs as independent entities, scheduling each job separately without considering their dependencies [4,5]. For CDP applications, the basic model employs two common strategies. The first combines all operations into a single "large" job submitted after all data are generated, causing resource idleness and job inefficiency. The second submits individual operations (that is, preprocessing and aggregate analysis) as separate jobs [6,7] but overlooks job dependencies, reducing overall application performance. Workflow models, particularly scientific workflows, define data dependencies and execution sequences among jobs within a complex workflow application [8–12]. However, they assume a fixed number and type of jobs at the time of workflow initiation, making them unsuitable for CDP applications, where data are generated incrementally. This results in a dynamic and unpredictable number of jobs, along with fluctuating resource demands driven by the varying volume and characteristics of accumulated data.

When CDP applications involve multiple jobs to process cumulative data in stages, efficient job scheduling is critical. Existing job scheduling strategies in clouds fall into three categories: rule-based, heuristic/metaheuristic-based, and machine learning-based approaches. Rule-based strategies, such as First-Come-First-Serve (HEFT) and Firstfit, struggle with dynamic resource availability, job constraints, and dependency management, resulting in lower resource utilization and execution efficiency [13–15]. Heuristic/metaheuristic approaches aim for optimal or near-optimal solutions in each scheduling round [16–25]. However, their lack of coordination for global optimization across rounds makes them unsuitable for long-running CDP applications, where data dynamically accumulate and jobs are irregularly generated and submitted over scheduling cycles. Machine learning-based approaches typically predict job resource demands to guide scheduling [26–29]. Although

these methods account for future demands, they do not learn from historical scheduling decisions to enhance future performance. Reinforcement learning, in contrast, marks a significant advancement in cloud job scheduling [30,31], as it optimizes decisions by learning from feedback on past performance [32,33]. However, existing models are not designed to accommodate the long-duration and dynamic natures of CDP applications. Such strategies designed for scientific workflows also fall short, as they fail to handle uncertain job arrivals and evolving execution progress [26,34–37].

To tackle these challenges, we propose CDP-EM, an innovative job execution model, and CDP-JS, a customized job scheduling strategy for CDP applications in multi-cloud environments. The CDP-EM model defines a CDP application as comprising multiple preprocessing jobs and an aggregate analysis job. By utilizing an execution agent and an application description model, preprocessing jobs are dynamically generated based on user-specified conditions, while the aggregate analysis job is automatically triggered upon the completion of all preprocessing jobs.

The CDP-JS strategy formulates the job scheduling problem for CDP applications in multi-cloud environments as a Markov Decision Process (MDP). It incorporates resource-level, job-level, and application-level information into the state space to capture resource supply–demand dynamics and the uncertainties of job generation, which impact application progress. The reward space is carefully designed to include multiple dimensions, including the quality of resource allocation, the distribution of intermediate data across multi-cloud data centers, the risk of Service Level Agreement (SLA) violations, and resource costs. These dimensions comprehensively evaluate the quality of scheduling decisions for CDP applications in multi-cloud environments.

To solve this decision-making process, CDP-JS employs the Proximal Policy Optimization (PPO) deep reinforcement learning algorithm. We carefully designed the actor–critic networks in our proposed PPO algorithm, which share the same neural network architecture to enhance training efficiency and stabilize policy updates in dynamic multi-cloud scenarios. The designed actor–critic networks adopt the integration of multiple neural network components to enable comprehensive feature extraction and fusion across the resource, job, and application-level states, and effectively learn complex scheduling patterns and generate precise scheduling decisions.

By integrating CDP-EM and CDP-JS, we provide a comprehensive scheduling solution for CDP applications in multi-cloud environments. The simulation results show that, compared to the state-of-the-art baselines described in detail in Section 6.3, our proposed solution can reduce the SLA violation rate and resource cost of CDP applications by an average of 34.8% and 23.4%, respectively. Real-world evaluations show average reductions of 27.2% and 31.3%, respectively. In summary, our contributions are as follows:

(1) We propose the first comprehensive scheduling solution tailored for CDP applications.
(2) We introduce the CDP-EM job execution model, supporting dynamic preprocessing job generation and automated, dependency-aware job execution.
(3) We propose the PPO-based CDP-JS job scheduling strategy, capturing progress dynamics from job uncertainty and optimizing scheduling via historical learning.
(4) We evaluate the proposed scheduling solution for CDP applications comprehensively using the real-world dataset and simulation experiments.

2. Related Works

2.1. Cloud Job Execution Model

Jobs are the basic unit of scheduling and execution for complex cloud applications. While some studies use terms like “task” or “instance”, this paper uniformly adopts “job”. Basic job execution models divide applications into independent, manageable job units,

assuming each job is autonomous and requires no state sharing or coordination. Job descriptions in these models thus focus on individual jobs' attributes, resource requirements, and execution conditions. Most works schedule and execute jobs based on a user-specified job's computational scale and resource demands, particularly CPU and memory demands [4,5]. Cai et al. identify six key criteria, including resource utilization, energy consumption, and load balancing, to constrain job execution conditions [6]. Zhang et al. decompose long-term continuous write applications (CWA) into continuously arriving jobs. The model assumes infinite execution, focusing on per-unit-time resource requirements of jobs while ignoring their relationships with parent applications [7].

Scientific workflows are the most representative job execution models for complex cloud applications. They define atomic jobs, specify job dependencies, and support both sequential and concurrent execution [12]. Workflow models encompass various job dependency patterns, including aggregation, distribution, parallelization, and pipeline processing [8]. Scientific workflows are often represented as Directed Acyclic Graphs (DAGs) or Directed Cyclic Graphs, with nodes as jobs and edges indicating dependencies or data flows. Taverna [9] modeled workflows using XML to define processors (executing jobs), data links (data flows), and runtime constraints, such as conditional branching and concurrency limits. Nextflow defines computational jobs as processes with input/output channels and resource requirements, while channels connect processes as data transfer units [10]. Wings used computational ontologies to model workflow constraints, detailing execution environments and resource needs [11]. However, existing scientific workflow models assume a fixed number of jobs within applications, making them unsuitable for CDP applications, where dynamic data accumulation causes constantly changing job quantities and resource demands.

2.2. Cloud Job Scheduling Strategy

Cloud job scheduling strategies are categorized into three types: rule-based, heuristic/metaheuristic-based, and machine learning-based.

Traditional rule-based scheduling strategies, such as First-Come-First-Served (FCFS) [13], Short-Job-First (SJF) [38], and Max–min [39], are widely used due to their simplicity, intuitiveness, and ease of implementation. However, in the face of the highly dynamic nature of resources and the complexity of job workloads in cloud computing environments, these deterministic strategies are gradually resulting in performance bottlenecks. To address this issue, recent works have proposed more sophisticated rules-based strategies. RADISH schedules jobs using the runtime load balancing status of the virtual machine (VM) as the criterion [14]. FDHEFT prioritizes tasks using fuzzy dominance sorting and then schedules them according to predefined rules. However, both strategies still fall short when dealing with scientific workflows that have complex job dependencies [15].

Heuristic/metaheuristic strategies are widely adopted in scientific workflow scheduling due to their powerful ability to find near-optimal solutions. Alam et al., Liu et al., and Meena et al. employ genetic algorithms to tackle cloud job scheduling challenges, with a focus on trust-aware job deployment across multiple clouds, data placement strategies in hybrid clouds, and cost-optimized job scheduling under deadline constraints, respectively [16–18]. Shi et al. integrate genetic algorithms with customized chromosome representations and fitness evaluations to simultaneously minimize cost and network latency in multi-cloud job scheduling [20]. Li et al. and Shi et al. propose job scheduling strategies based on the particle swarm optimization (PSO) algorithm [19,21]. PSO-MC, which integrates particle swarm optimization with membrane computing, effectively reduces the execution time and cost of cloud jobs [19]. AdPSO enhances the particle swarm algorithm through an adaptive inertia weight strategy, improving job execution efficiency

and resource utilization [21]. Hybrid metaheuristic algorithms, such as PPTS-PSO [22], DQ-HEFT [23], DB-ACO [24], and HBABC [25], combine the strengths of different approaches to improve adaptability and flexibility in job scheduling. Although heuristic-based strategies can obtain near-optimal scheduling solutions, they struggle to adapt to the dynamic variation in resource supply and demand in a multi-cloud environment, often resulting in "myopic" decisions across multiple rounds of scheduling.

Machine learning-based strategies offer a promising approach for solving or optimizing scheduling problems. These strategies typically employ deep learning models to predict the resource consumption or execution time for cloud jobs, thereby guiding future scheduling decisions. Tian et al. utilize LSTM to predict future resource demands of cloud jobs, addressing workload fluctuations [26]. Mao et al. employ scalable graph neural network embeddings to extract workflow job features, guiding scheduling decisions [27]. Wang et al. integrate naive Bayes to learn probabilistic relationships between job features and execution times, optimizing job order and resource allocation [28]. Hu et al. classify jobs using fuzzy clustering, predict resource demands per group, and design scheduling strategies to enhance job efficiency and resource utilization [29].

Reinforcement learning (RL), particularly deep reinforcement learning (DRL), has demonstrated significant potential in complex cloud application scheduling due to its powerful decision-making and adaptability to dynamic environments [32,33]. QIN et al. apply RL to deadline-constrained scientific workflow scheduling, where the DCMORL algorithm reduces execution costs and energy consumption [30]. Tian et al. aimed to minimize the average response time and maximize resource utilization by dynamically selecting scheduling rules at each scheduling point for online jobs [26]. DRL integrates deep learning's perception capabilities with RL's decision-making, enabling solutions for resource scheduling in complex and dynamic environments [31]. Ran et al. formulated job scheduling as a constrained dynamic optimization problem and used the Deep Deterministic Policy Gradient (DDPG) network to find optimal job allocation solutions that met SLA requirements [34]. Chen et al. propose a Dueling Double Deep Q-Network (D3QN)-based collaborative scheduling method with an adaptive time-step scheduling mechanism, addressing high-dimensional objective optimization for heterogeneous workflows under continuity constraints in cloud computing [35]. Zhang et al. propose a workflow partitioning and scheduling framework based on an enhanced DQN algorithm, which optimizes temporal efficiency and load balancing for IoT applications in multi-cloud environments by minimizing data transfer overheads [36]. Mangalampalli et al. designed a task partitioning and scheduling mechanism using an improved asynchronous A3C algorithm, achieving a joint reduction in makespan and energy consumption through subtask granularity optimization [37]. Although effective for general application scheduling, these strategies lack specific models for CDP applications, where data are generated incrementally, resulting in a dynamic job landscape and fluctuating resource demands driven by the varying volume and characteristics of the accumulated data. Further research is needed to address these unique characteristics.

3. Problem Definition

3.1. Cumulative Data Processing Application

CDP applications handle data that are incrementally generated over a long period. The data processing in these applications consists of two stages: the preprocessing stage and the aggregate analysis stage. In the preprocessing stage, operations such as transformation, statistical analysis, and filtering are applied to the incrementally generated data, while

the aggregate analysis stage involves complex mining and analysis operations on all the preprocessed data. A CDP application can be formulated as:

$$CDP = (PreOps,\ AggOps,\ DataSource, DStartTime, DEndTime,\ Deadline)$$

Here, $PreOps$ and $AggOps$ represent the operations performed during the preprocessing and aggregate analysis stages, respectively. $DataSource$ denotes the application's data source, while $DStartTime$ and $DEndTime$ specify the start and end times of the data generation. $Deadline$ indicates the user-specified completion deadline for the CDP application.

Due to dependencies between the preprocessing and aggregate analysis stages, the duration of a CDP application can be expressed as:

$$T\_cdp = duration^{pre} + duration^{agg} \tag{1}$$

Here, $duration^{pre}$ and $duration^{agg}$ represent the durations of the preprocessing and aggregate analysis stages, respectively. The service quality of a CDP application, denoted as SLA, can be measured by whether it is completed before the deadline.

$$SLA = \begin{cases} 1, & Starttime + T\_cdp \leq Deadline \\ 0, & else \end{cases} \tag{2}$$

where $Starttime$ is the application's start time.

3.2. Multi-Cloud Environment

A multi-cloud environment consists of multi-cloud data centers, denoted by $DCS = \{dc_1, dc_2, \ldots, dc_n\}$. Each data center dc_i contains a set of physical servers where $machine_{i,j}$ denotes the j-th server in the i-th data center. Each dc_i can provide tn_i types of VM images, defined as $VTS_i = \left\{vtype_i^1, vtype_i^2, \cdots, vtype_i^{tn_i}\right\}$. In this paper, we focus on the allocation of CPU and memory resources for VMs. Therefore, each $vtype_i^k$ can be defined as $vtype_i^k = \left(vCPUCapacity_i^k, vMemCapacity_i^k, vprice_i^k\right)$. Where $vCPUCapacity_i^k$, $vMemCapacity_i^k$, and $vprice_i^k$ represent the CPU capacity, the memory size, and the rental price per-unit time of the j-th VM type in dc_i, respectively.

In a multi-cloud environment, we assume that the data transfer time between VMs within the same data center is negligible (i.e., zero.). However, the data transmission time between VMs located in different data centers is given by:

$$t\_trans\left(dc_p, dc_q, data\right) = \frac{sizeof(data)}{bd\left(dc_p, dc_q\right)} \tag{3}$$

where $bd\left(dc_p, dc_q\right)$ denotes the network bandwidth between the data centers dc_p and dc_q, and $sizeof(data)$ denotes the size of the transmitted data. Specifically, $bd\left(dc_p, Client\right)$ refers to the network bandwidth between a data center and the client.

3.3. Resource Cost

For a CDP application, its resource cost can be expressed as follows:

$$Cost\_cdp = vm\_cost + data\_cost \tag{4}$$

where vm_cost represents the cost of renting VM resources during the preprocessing and aggregation analysis stages, while $data\_cost$ denotes the cost of the data transfer between these operations across multiple data centers. Based on the billing policies of the cloud data centers, the cost of data uploading by users to the centers is not included.

3.4. Problem Definition

Let M CDP applications run in a multi-cloud environment consisting of K data centers, and any dc_i contains n_i physical servers. The scheduling problem for CDP applications can be formulated as follows:

$$\begin{cases} Max \sum_{i=1}^{M} SLA(CDP_i), \\ Min \sum_{i=1}^{M} cost(CDP_i) \end{cases} \tag{5}$$

Subject to:

$$\forall t, machine_{i,j}, \sum_{vm_k \in VA(machine_{i,j},t)} CPUAlloc(vm_k) \leq CPUTotal(machine_{i,j}) \; i \in [1,K], j \in [1,n_i] \tag{6}$$

$$\forall t, machine_{i,j}, \sum_{vm_k \in VA(machine_{i,j},t)} MemAlloc(vm_k) \leq MemTotal(machine_{i,j}) \; i \in [1,K], j \in [1,n_i] \tag{7}$$

where $VA(\cdot)$ represents the set of VM instances deployed on a physical server at time t. The $CPUAlloc(\cdot)$ and $MemAlloc(\cdot)$ functions indicate the CPU and memory resources allocated to VM instances, respectively; the $CPUTotal(\cdot)$ and $MemTotal(\cdot)$ represent the total CPU and memory resources of the physical server.

The scheduling objectives are twofold: maximizing the number of applications that meet their deadline constraints and minimizing the resource cost of application execution. The scheduling constraint ensures that the cumulative resource allocation of VMs deployed on any physical server at any time does not exceed the server's total available resources.

4. Job Execution Model for CDP Applications

4.1. CDP-EM Model

The framework of the CDP-EM job execution model is illustrated in Figure 2. In the CDP-EM model, the preprocessing stage of a CDP application involves multiple data preprocessing jobs, whereas the aggregate analysis stage is handled by a single aggregate analysis job. An application agent is assigned to each CDP application, automating the generation and submission of both job types.

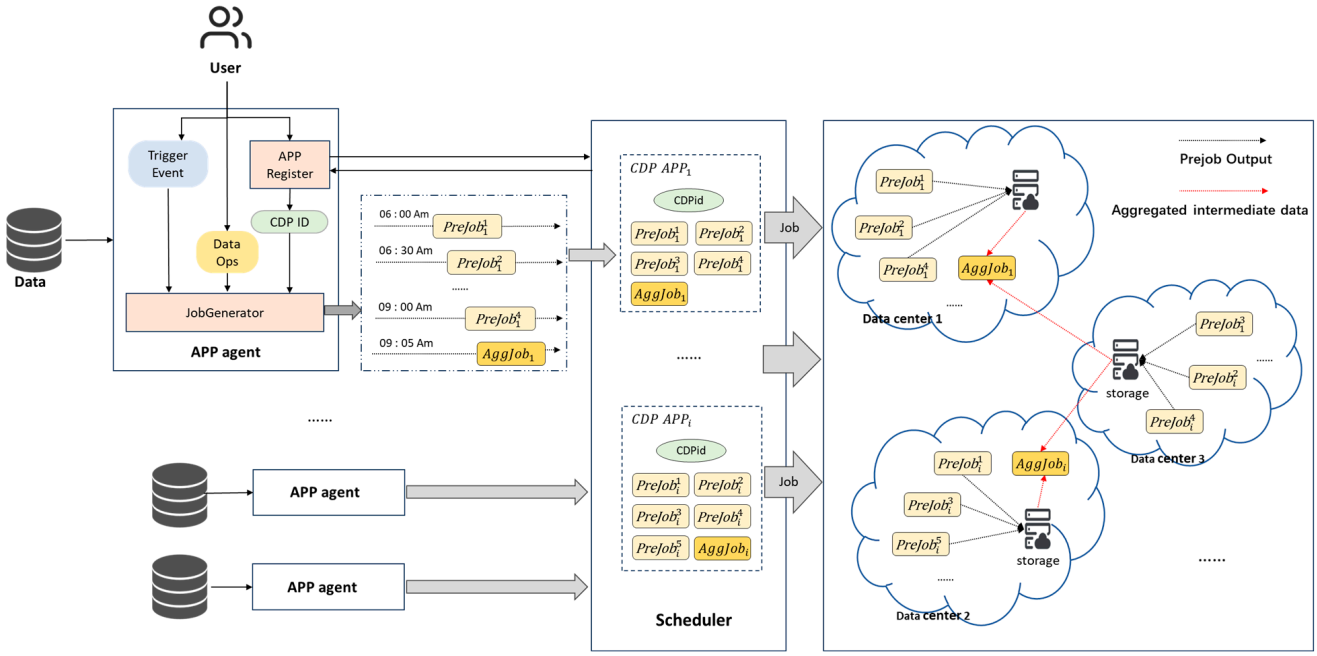


Figure 2. Framework of CDP-EM model.

In the CDP-EM model, the application and job are described as follows:

$$CDP = (CDPid, PreJobs, AggJob, Deadline)$$

$$PreJobs = (CDPid, PreOps, DataSource, DStartTime, DEndTime, VMReq, VMNum, LocDC, TrigerEvent)$$

$$AggJob = (CDPid, AggOps, VMReq, VMNum, LocDC)$$

where $CDPid$ is the unique ID of a CDP application, which is obtained when the application agent submits a CDP application creation request to the cloud scheduler; the $CDPid$ is then used to establish a clear association between the application and its constituent jobs during subsequent job submissions.

Preprocessing and aggregate analysis jobs specify their resource requirements as VM instances, including resource configuration $VMReq$ and instance count $VMNum$. $VMReq$ defines the job's CPU and memory needs per VM as $VMReq = (vCPUReq, vMemReq)$. Notably, resource requirements may vary between preprocessing and aggregate analysis jobs.

In the CDP-EM model, the application agent interprets the trigger conditions specified in $TrigerEvent$ within $PreJobs$. During the data generation period, from the start time $DStartTime$ to the end time $DEndTime$, when the trigger conditions are satisfied, the application agent fetches newly accumulated data from the data source and automatically generates and submits a preprocessing job based on user-specified operations $PreOps$. These trigger conditions can include periodic submissions, thresholds for data accumulation, etc.

In the CDP-EM model, the cloud scheduler allocates computing resources within the same data center for each job, storing intermediate data locally to reduce communication overhead and improve efficiency. Preprocessing jobs belonging to a CDP application can be distributed across different data centers and executed in a different order from their submission so as to maximize resource utilization in a multi-cloud environment. After the data generation period, the application agent submits an aggregate analysis job based on $AggOps$. The cloud scheduler schedules it once all preprocessing jobs are completed, pulling intermediate results from their respective data centers to produce the final output.

4.2. Problem Redefinition

Based on the CDP-EM model, a CDP application is composed of a set of cloud jobs and can be expressed as follows:

$$CDP\_JOBS = \{PreJob_1, PreJob_2, \cdots, PreJob_n, AggJob\}$$

Based on the CDP-EM model, we can determine the specific calculation formulas for the variables $duration^{pre}$ and $duration^{agg}$ in Equation (1), thereby quantifying the total execution time of a CDP application T_cdp. Additionally, the calculation formulas for the variables in Equation (4) can be established to measure the resource cost for executing a CDP application $Cost\_cdp$.

4.2.1. Calculation of $duration^{pre}$

Let the preprocessing job $PreJob_i$ have an input data size of $datasize_i$ and a computational load of $length_i$. Denoting the total number of VM instances allocated to the job

as $vnum_i$, and the computational capacity per instance as $vflops_i$, the turnaround time of $PreJob_i$ can be calculated as follows:

$$TA(PreJob_i) = t\_Wait(PreJob_i) + t\_DTrans(PreJob_i) + t\_Exe(PreJob_i) \tag{8}$$

Here, $t\_Wait(\cdot)$ is a job's waiting time for scheduling, and $t\_DTrans(\cdot)$ is its data transmission time calculated using Equation (3) based on the network bandwidth between the application agent and the hosting cloud data center. $t\_Exe(\cdot)$ is its execution time and can be roughly calculated as follows:

$$t\_Exe(PreJob_i) = \frac{length_i.}{vflops_i \times vnum_i} \tag{9}$$

The end time of $PreJob_i$ can then be expressed as:

$$FT(PreJob_i) = ST(PreJob_i) + TA(PreJob_i) \tag{10}$$

where $ST(\cdot)$ is the job's submission time. Since the preprocessing stage involves multiple dynamically generated jobs, its $duration^{pre}$ can be expressed as follows:

$$duration^{pre} = \max_{PreJob_i \in CDP\_JOBS} FT(PreJob_i) - \min_{PreJob_i \in CDP\_JOBS} ST(PreJob_i) \tag{11}$$

4.2.2. Calculation of $duration^{agg}$

Due to the data dependency, the submission time of the aggregate analysis job is the maximum of all preprocessing jobs' end times. $duration^{agg}$ is then equal to the turnaround time of the aggregate analysis job, expressed as:

$$duration^{agg} = TA(AggJob) \tag{12}$$

$$TA(AggJob) = t\_Wait(AggJob) + t\_DTrans(AggJob) + t\_Exe(AggJob) \tag{13}$$

The execution time is calculated similarly to the preprocessing job. However, since the aggregate analysis job must receive intermediate data from multiple preprocessing jobs, the data transmission time is:

$$t\_DTrans(AggJob) = \max_{PreJob_i \in CDP\_JOBS} (t\_trans(dc(PreJob_i), dc(AggJob), idatasize_i)) \tag{14}$$

where $dc(\cdot)$ indicates the job's hosting data center, and $t\_trans(\cdot)$ is the same as Equation (3). $idatasize_i$ represents intermediate data size of $PreJob_i$.

4.2.3. Calculation of *Cost_cdp*

The total resource cost of a CDP application includes costs that occurred in both the preprocessing stage and the aggregate analysis stage. We denote them as *p_cost* and *a_cost*, respectively. Referring to Equation (4), *p_cost* mainly considers the VM rental cost, while *a_cost* includes both the VM rental cost and the intermediate data transferring cost.

The VM rental cost for both preprocessing and aggregate analysis jobs can be defined as follows:

$$j\_vmcost = vnum \times consumetime \times vprice \tag{15}$$

where *vnum* is the number of allocated VM instances, *vprice* is the per-unit time price of a VM instance, and *consumetime* is the sum of the job's data transmission and execution time, referring to Equations (8) and (13).

The data transmission cost of the aggregate analysis job can be calculated as follows:

$$data\_cost = \sum_{PreJob_i \in CDP\_JOBS} idatasize_i \times dprice_i \quad (16)$$

Here, $dprice_i$ denotes the price of the data transmission per unit size between the data centers where $PreJob_i$ and $AggJob$ are located. If the job pair is located in the same data center, the data transmission cost is assumed to be zero [40,41].

In summary, the cost of the preprocessing stage, the cost of the aggregate analysis stage, and the total cost of a CDP application can be expressed as follows, respectively:

$$p\_cost = \sum_{PreJob_i \in CDP\_JOBS} j\_vmcost(PreJob_i) \quad (17)$$

$$a\_cost = j\_vmcost(AggJob) + data\_cost \quad (18)$$

$$Cost\_cdp = p\_cost + a\_cost \quad (19)$$

5. Job Scheduling Strategy for CDP Applications

A CDP application comprises multiple preprocessing jobs and a final aggregate analysis job, typically running for extended periods. Consequently, the job scheduling for the CDP application spans multiple scheduling rounds in a multi-cloud environment, with the application's final execution efficiency determined by a combination of these sequential scheduling decisions. In addition, since multiple preprocessing jobs share similar resource usage patterns, feedback from previous scheduling can inform future decisions. Therefore, we propose a CDP-JS scheduling strategy, which models the job scheduling of the CDP application as a Markov Decision Process and applies deep reinforcement learning to derive an optimal solution.

5.1. Markov Decision Process Modeling

5.1.1. State Space

In this study, we assume that during the application scheduling process, resource prices remain stable but vary across data centers, with physical servers within each data center being homogeneous. For the job scheduling of CDP applications, the state space must account for both resource and job states across multiple cloud data centers.

1. Cloud Resource State

Cloud data centers allocate resources to CDP applications in the form of VMs, with multiple VM instances deployed on the same physical server, sharing server resources. Therefore, the defined cloud resource state includes the VM state and the physical server state. This can be represented as follows:

$$State\_MC = (State\_VM, State\_PS)$$

$State\_VM$ captures the resource configuration and runtime utilization of VM instances across multi-cloud data centers. To manage VM deployment efficiently, we limit the maximum number of VM instances per physical server to 10. The state of each VM instance is described by a quintuple:

$$vm\_state = (vCPUutil, vMemutil, vCPUcapacity, VMemcapacity, consumetime)$$

where $vCPUutil$ represents the ratio of the actual CPU capacity utilized by the running job to the total CPU capacity of the VM instance. Similarly, $vMemutil$ indicates the percentage of memory occupied by the job within the VM instance. The resources allocated to the instance are determined by its VM type, $vtype$, as described in Section 3.2. Furthermore,

consumetime, as illustrated in Section 4.2, represents the amount of time the instance occupies these resources.

Based on the state of each VM instance, the VM state of the multi-cloud data centers, $State\_VM$, is represented as a matrix formed by sequentially concatenating the states of all VM instances in data centers, as shown in Figure 3a. In the $State\_VM$ matrix, VM instance states are arranged in the order of the data center and physical server. For each physical server, matrix rows are allocated based on the maximum number of deployable VM instances. If the actual number of deployed instances is less than the maximum, the remaining rows are padded with zeros.

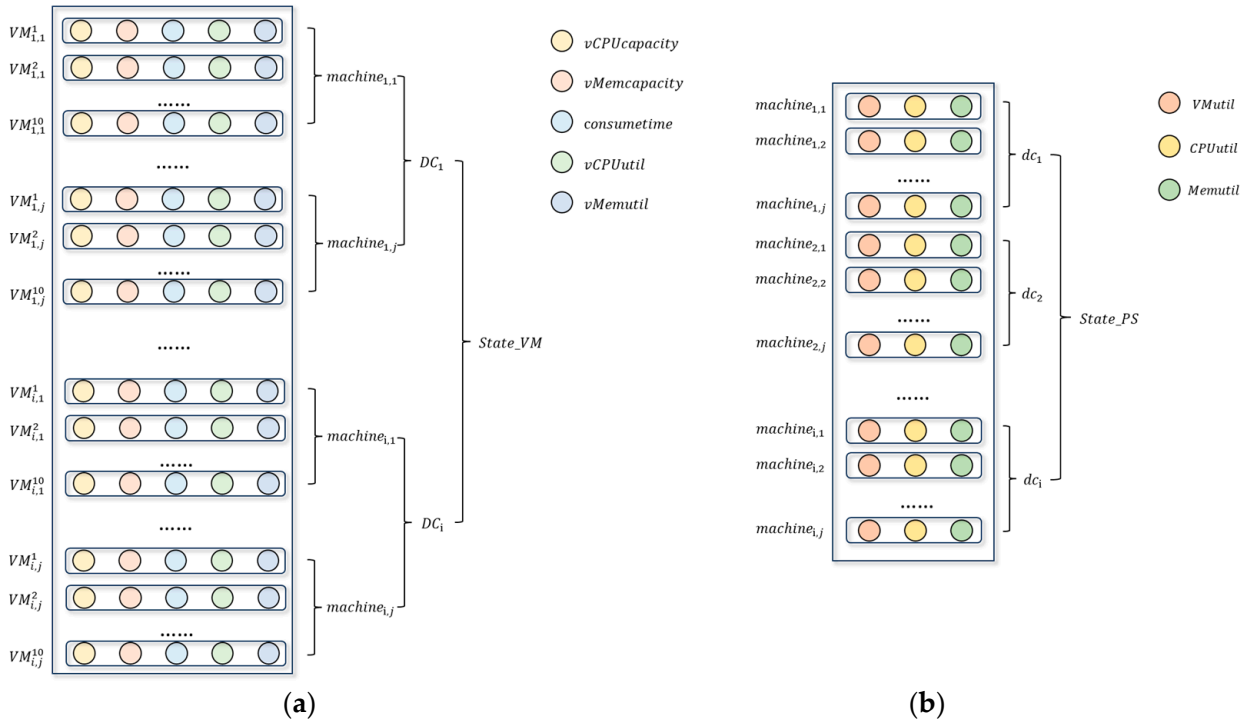


Figure 3. Example of the cloud resource state: (**a**) state of VMs and (**b**) state of physical servers.

$State\_PS$ represents the resource allocation states of physical servers, expressed as an $N \times 3$ matrix, as shown in Figure 3b. Each matrix row describes the state of a physical server with three dimensions.

$$ps\_state = (VMutil, CPUutil, Memutil)$$

$VMutil$ is defined as the ratio of the number of deployed VM instances to the total available instance slots on the physical server. Similarly, $CPUutil$ and $Memutil$ represent the occupation ratios of the CPU and memory resources by all deployed VM instances on the physical server, respectively. These ratios indicate the proportion of allocated CPU and memory resources for the VMs relative to the total available resources of the physical server.

2. Pending Job State

The pending job state refers to the state of jobs that have been submitted but not yet successfully scheduled at a scheduling decision point in a multi-cloud environment. Assuming there are N pending jobs, the pending job state, denoted as $State\_PendingJob$, is represented by an $N \times 9$ matrix, as shown in Figure 4, where each row corresponds to the state of an individual pending job. All pending jobs' states are recorded in the rows of the matrix in the order of their arrival time. Specifically, the state of a pending job is

expressed by a nine-dimensional tuple, which includes both the job-level and application-level information.

$$pendingjob\_state = (jobtype,\ length, datasize, vnum, vCPUReq, vMemReq, density, app\_progress, app\_perjob)$$

where $jobtype$, $length$, and$datasize$ denote the type, the computational load, and the input data size of the job; $vnum, vCPUReq$, and$vMemReq$ express the required number and resource configuration of VM instances for the job; $density$ indicates the job's computational intensity, calculated as $density = length/remaintime$, where $remaintime$ represents the time remaining until the application deadline; $app\_progress$ denotes the normalized time to a deadline of the CDP application to which the job belongs; and app_perjob expresses the expected progress of job execution of the CDP application, which can be expressed as follows:

$$app\_progress = \frac{Deadline - Curtime}{Deadline - Starttime} \tag{20}$$

$$app\_perjob = \frac{num\_remainingjob}{Deadline - Curtime} \tag{21}$$

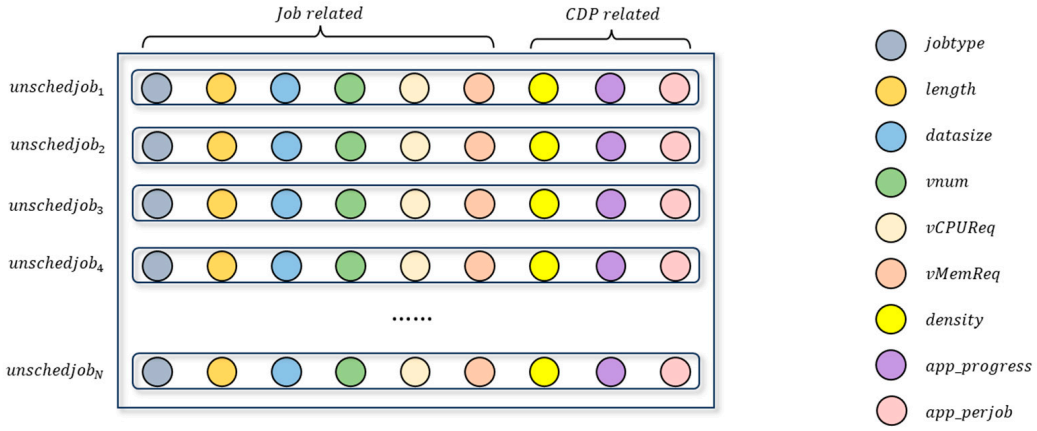


Figure 4. Example of the pending job state.

Here, $Curtime$ denotes the current time, and $num\_remainingjob$ denotes the number of pending jobs for the application in the job queue.

In sum, the first seven dimensions in $pendingjob\_state$ express the job-level state for a pending job, while the $app\_progress$ and app_perjob dimensions represent the state of the CDP application that the pending job belongs to. In addition to expressing the resource requirements and computational characteristics of the pending jobs, the three dimensions-$density$, $app\_progress$, and app_perjob in $pendingjob\_state$ implicitly represent the urgency with which the job, along with all jobs in its parent application, should be scheduled to guarantee the application's SLA, thereby enabling the informed scheduling decisions.

3. Historical Job Demand State

Given the long-term nature of CDP applications and the repetitive patterns of preprocessing jobs, analyzing historical job characteristics and resource demands enables effective predictions of near-future resource needs. This supports the cloud scheduler in balancing current and incoming job demands, enabling decisions with lasting optimization effects.

In our design, the historical job demand state denoted as $State\_Hisreq$ is modeled as a time series, as shown in Figure 5. Demand statistics for all jobs in the historical job queue are recorded at 30 s intervals. The state for each interval includes five statistical features: the total input data size, $datasize\_sum_t$; total computational load, $length\_sum_t$; total number of required VM instances, $vnum\_sum_t$; total CPU capacity of the required VM instances,

$vCPUReq\_sum_t$; and the total memory size of required VM instances, $vMemReq\_sum_t$. The 30 s interval balances data sparsity and information retention effectively.

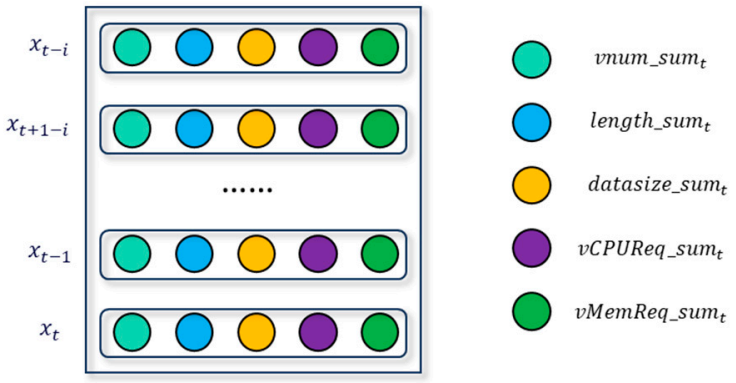


Figure 5. Example of the historical job demand state.

5.1.2. Action Space

Action represents the decision based on the current state in a Markov Decision Process, linking the present state to future states and key to describing the system's dynamics. In the CDP-JS model, the scheduler selects one or more jobs from the pending job queue and determines the data center in which they will run, as well as the type of VM to be allocated. To reduce action space complexity, our proposed scheduling strategy traverses the job queue in order of job arrival time in each scheduling round, sequentially determining the scheduling decision for each job. Consequently, the action space is defined as follows:

$$action = \left\{none, allocvt_1^1, allocvt_1^2, \cdots, allocvt_n^{tn_n}\right\}$$

where *none* represents that the job is not scheduled, while $allocvt_i^j$ represents that the job is scheduled to dc_i, with $vtype_i^j$ assigned. The CPU capacity and memory size of $vtype_i^j$ should be equal to or higher than the job's VM requirement.

Based on the action decision, the cloud scheduler deploys VM instances for the target scheduled job in the selected data center, according to the chosen VM type. If the remaining available resources in the selected data center meet the job's resource requirements, the scheduler deploys VM instances on the physical servers within that data center based on the load balancing criterion. Otherwise, the scheduling attempt fails.

5.1.3. Rewards

In Markov Decision Process modeling, rewards evaluate the feedback from actions taken in a given state. Setting an appropriate reward mechanism is essential for optimizing decision-making and enhancing final performance. For job scheduling in long-term CDP applications, the primary goal is to meet the application SLA by ensuring completion before the specified deadline while minimizing execution costs. To this end, we define five reward components as follows: the quality of VM resource allocation, intermediate data distribution, application SLA violation risks, cost of the target scheduled job, and the penalty for unsuccessful scheduling. The reward for any scheduling decision is the sum of these five components, all normalized to the same dimension.

1. Quality of VM Resource Allocation

The quality of VM resource allocation determines the execution efficiency of a CDP application. For the target scheduled job, the VM resource allocation quality is defined as:

$$R_{vm} = \begin{cases} \alpha \times \frac{vCPUAlloc}{vCPUReq} / \frac{vCPUMax}{vCPUMin} + \beta \times \frac{vMemAlloc}{vMemReq} / \frac{vMemMax}{vMemMin}, \\ \qquad vCPUAlloc \geq vCPUReq \; and \; vMemAlloc \geq vCPUReq \\ 0, \qquad else \end{cases} \tag{22}$$

where $vCPUMax$, $vMemMax$, $vCPUMin$, and $vMemMin$ represent the maximum and minimum CPU and memory capacities across all VM types in multiple data centers, respectively. $vCPUAlloc$ and $vMemAlloc$ denote the CPU and memory of the VM instance allocated to the job. The weighting factors α are β sum to 1. A higher quality of VM resource allocation yields a greater reward while failing to meet the job's resource requirements results in a reward of zero.

2. Intermediate Data Distribution

When jobs of a CDP application are distributed across multiple data centers, the transmission of intermediate data between preprocessing and aggregate analysis jobs significantly impacts the application's performance. The intermediate data distribution reward reflects the data transmission overheads resulting from these job distributions, which is defined as follows:

$$R_{data} = \begin{cases} \frac{1}{m} \sum_{j=1}^{m} nor\_bd\left(dc(PreJob_i), dc\left(PreJob\_f_i^j\right)\right), \; if \; scheduling \; preprocessing \; job \\ \frac{1}{n} \sum_{j=1}^{n} nor\_bd\left(dc\left(PreJob_j\right), dc(AggJob)\right), \quad if \; scheduling \; aggregate \; analysis \; job \end{cases} \tag{23}$$

where $PreJob\_f_i^j$ represents the preprocessing jobs of the same CDP application as the target scheduled job $PreJob_i$, which have been completed or are running, and m denotes the total number of these jobs. And $dc(\cdot)$ denotes the data center where a job resides. $nor\_bd(\cdot)$ denotes the normalized network bandwidth between two data centers. For two jobs located in the same data center, $nor\_bd(\cdot)$ is set to a value greater than 1.

As described above, when the target scheduled job is an aggregate analysis one, the intermediate data distribution of its corresponding CDP application is represented by the average of the normalized network bandwidth between the application's aggregate analysis job and preprocessing jobs. A higher bandwidth indicates better data distribution, resulting in a higher reward. When the scheduled job is a preprocessing job, and the corresponding aggregate analysis job has not been scheduled, we use the average of the normalized network bandwidth between this job and all scheduled preprocessing jobs within the same application as the expected intermediate data distribution.

3. Application SLA Violation Risk

Application SLA violation risk is a negative reward mechanism that penalizes failure to meet application deadlines, encouraging optimal scheduling decisions to avoid SLA violations. The application SLA violation risk reward is defined as follows:

$$R_{risk} = \begin{cases} -\frac{1}{N} \sum_{i=1}^{N} \left(1 - \frac{Deadline_i - t}{\text{t\_exp}\prime(CDP\_JOBS_i)}\right), & t + TA\_exp(CDP\_JOBS_i) > Deadline_i \\ 0, & else \end{cases} \tag{24}$$

where $TA\_exp(CDP\_JOBS_i)$ is the expected turnaround time for unscheduled jobs of a CDP_i application in the job queue, with N being the total number of generated CDP applications and t being the current time. $TA\_exp(CDP\_JOBS_i)$ is estimated based on the turnaround times of the already scheduled jobs within CDP_i. Specifically, the average

turnaround time of the scheduled jobs is calculated and multiplied by the total number of unscheduled jobs. Clearly, more unscheduled jobs and longer turnaround times before the deadline lead to a higher application SLA violation risk and a larger negative reward.

4. Cost of Target Scheduled Job

The job cost is a negative reward mechanism that represents the resource cost incurred by the target scheduled job. The cost of the preprocessing job, $j\_vcost(PreJob_i)$, and that of the aggregate analysis job, a_cost, are calculated according to formulas (15) and (18), respectively. The reward can be expressed as follows:

$$R_{cost} = \begin{cases} -nor\_vc(j\_vcost(PreJob_i)), & scheduling\ preprocessing\ job \\ -nor\_vc(a\_cost), & scheduling\ aggregate\ analysis\ job \end{cases} \quad (25)$$

where $nor\_vc(\cdot)$ denotes the normalized cost of the target scheduled job, which is the ratio of the job's resource cost to the maximum cost of all the scheduled jobs.

5. Penalty for Unsuccessful Scheduling

When designing the reward function, it is necessary to consider the situation of unsuccessful scheduling actions and set corresponding penalties for them. Two scenarios need to be considered: one is when the allocated VM type does not meet the resource requirements of the job, and the other is when the available resources in the allocated data center are less than the total resource requirements of the job. If the scheduling action results in either of these two situations, a negative reward will be imposed as a penalty.

The penalty reward can be expressed as:

$$R_{sch} = \begin{cases} -k & , \quad if\ condition\ 1\ or\ 2 \\ 0 & , \quad else \end{cases} \quad (26)$$

where k is the penalty coefficient. If the scheduling action conforms to the above two unsuccessful scenarios, the penalty reward is set as a negative value, and all other reward components as set to zero. Otherwise, the penalty reward is set to zero. Typically, the value of k is several times larger than the maximum absolute value of the other reward components, thereby significantly guiding the decision-making process away from insufficient scheduling decisions. In our design, we set this multiplier to three.

5.2. PPO-Based Job Scheduling

Deep reinforcement learning (DRL) solves complex Markov Decision Processes (MDPs) by learning optimal policies without explicit programming. Proximal Policy Optimization (PPO) is an important algorithm in DRL, which improves learning stability and efficiency by limiting the extent of policy updates and balancing exploration with exploitation [42]. Compared to other DRL algorithms like DQN, PPO offers higher computational efficiency and stability, avoids the high variance problem, and does not rely on experience replay like DQN, making it more effective for handling high-dimensional and complex tasks [43].

In multi-cloud job scheduling, the PPO algorithm offers unique advantages. It enables real-time decision-making based on dynamically changing resource conditions, adapting to variable environments while ensuring stability. PPO maintains good convergence and robustness through balanced exploration and exploitation. Using the actor–critic architecture, the actor network maps states to optimal actions, and the critic network evaluates action quality via the value function. The PPO clipping mechanism prevents excessive policy updates, ensuring robust and adaptive decision-making in complex, uncertain multi-cloud resource allocations. Thus, PPO is suitable for efficient decision optimization in dynamic

multi-cloud job scheduling environments. We selected PPO as the core algorithm of our scheduling solution.

5.2.1. Actor and Critic Network Design

In our design, the actor and critic networks share the same overall architecture, with the only difference being that the output layer of the actor network incorporates a softmax layer to accommodate the discrete action space in cloud job scheduling. We, therefore, present the detailed structure of the actor network in Figure 6. All states defined in the state space, including the VM state (*State_VM*), the physical server state (*State_PS*), the pending job state (*State_PendingJob*), and the historical job demand state (*State_Hisreq*), are used as inputs to the actor network. The VM_Autoencoder component extracts latent features from *State_VM* and reduces dimensionality via the autoencoder (*Latent VM encoding*). The Concat component then concatenates the *Latent VM encoding* with *State_PS* and *State_PendingJob*, capturing the job and resource states at the current scheduling time across multiple cloud data centers, thereby generating a comprehensive feature representation. The representation is passed to the DC_Encoder component, which produces a latent feature expression (*Latent DC encoding*) for the multi-cloud data centers. The Req_Predictor component uses *State_Hisreq* to predict the resource requirements and data processing scale for upcoming jobs (*Predicted Req*). In the final output component, Act_Dec, the actor network generates a probability distribution over actions using the *Latent DC encoding* and *Predicted Req*, while the critic network outputs a scalar value representing the estimated state value.

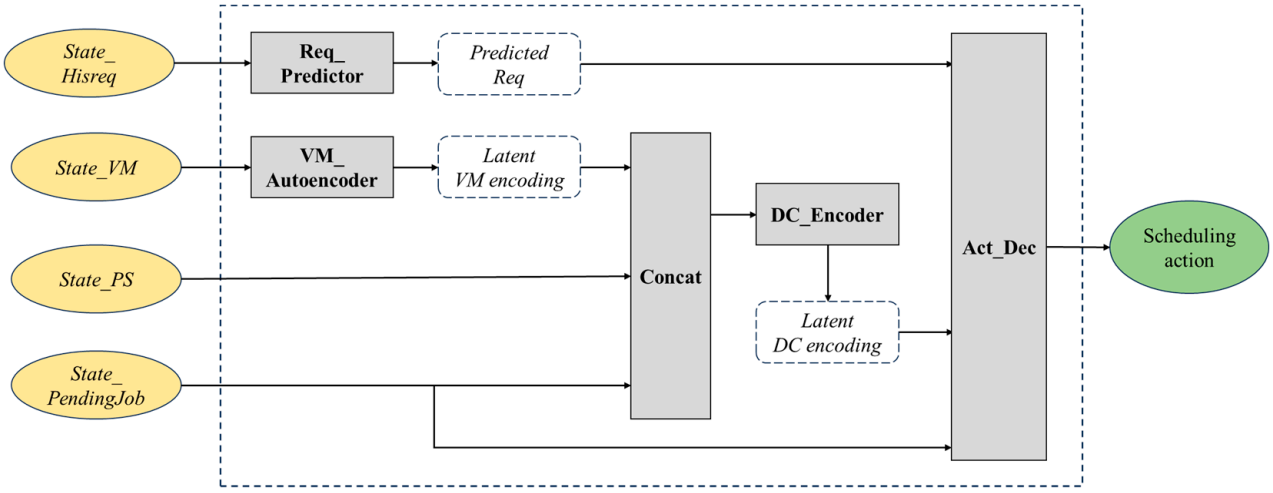


Figure 6. The architecture of actor and critic networks in CDP-JS.

1. VM_Autoencoder

In multi-cloud data centers, the number of VMs is vast, making the *State_VM* matrix significantly larger than the other three state matrices. To address this, the VM_Autoencoder component employs an autoencoder to extract key features from *State_VM* and reduces its dimensionality.

To preserve the deployment relationship between VMs and physical servers, we divide the *State_VM* matrix into sub-matrices, each corresponding to the VM state on an individual physical server. Each sub-matrix is encoded by the VM_Autoencoder's encoder into a five-dimensional latent feature vector. These vectors are then concatenated to form the *Latent VM encoding*, representing the overall VM states across the multi-cloud data centers, as shown in Figure 7.

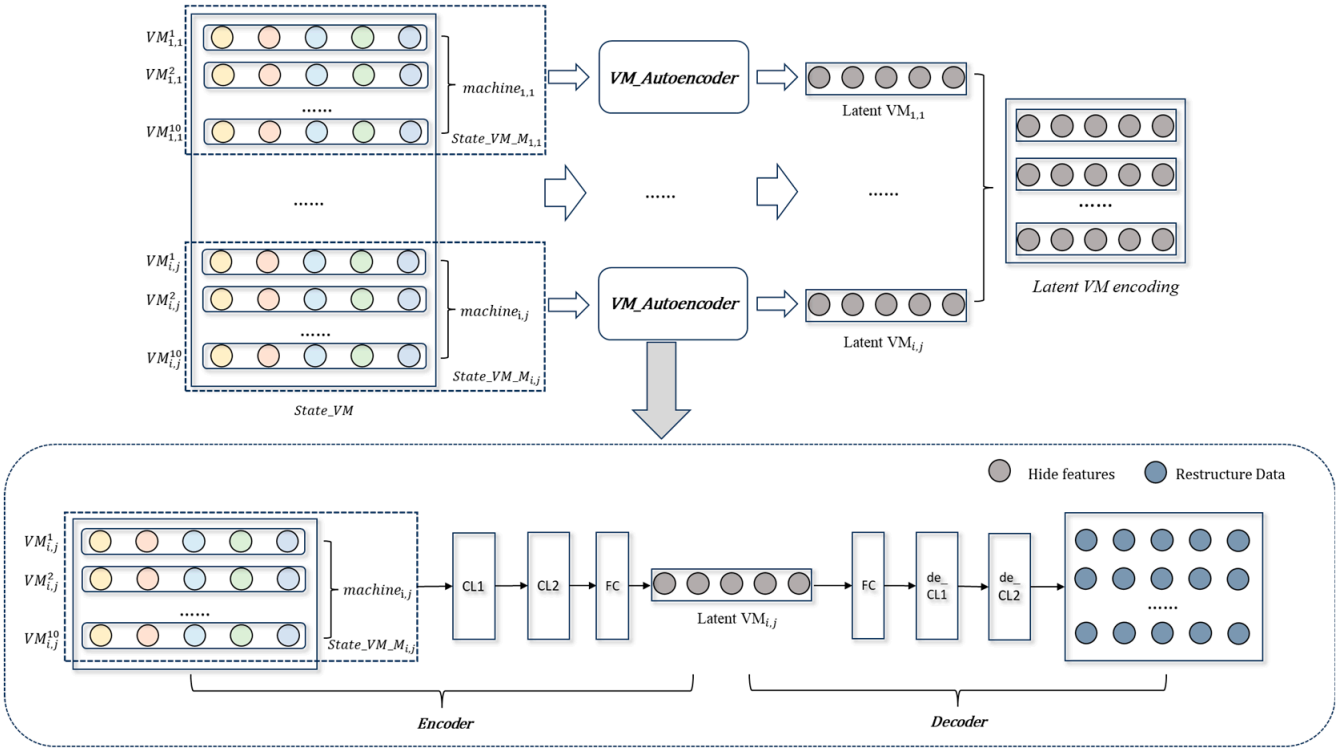


Figure 7. The architecture of the VM_Autoencoder.

The encoder of the VM autoencoder consists of two convolutional layers followed by a fully connected layer. The convolutional layers use 1×3 kernels with a stride of 1 and 16 output channels, employing ReLU activation to enhance the network's nonlinear capabilities. After the convolutional operations, the feature matrix is flattened into a one-dimensional vector, which is then processed by the fully connected layer to extract a five-element feature vector.

The decoder takes this feature vector as input, processes it through a fully connected layer with 160 neurons, and reconstructs the feature matrix using two deconvolution layers. The parameters of the deconvolution layers mirror those of the corresponding convolutional layers in the encoder to ensure accurate information reconstruction, with ReLU as the activation function.

2. Concat

The Concat component is responsible for concatenating the *Latent VM encoding*, *State_PS*, and *State_PendingJob* to form the complete state matrix of the multi-cloud data centers, which is used as the input for the DC_Encoder component to extract the comprehensive latent features at the multi-cloud level.

As shown in Figure 8, in our design, the *Latent VM encoding* matrix is first horizontally concatenated with the *State_PS* matrix. In the resulting concatenated matrix, each row represents the combined features of a physical server and all its associated VMs. Then, this concatenated matrix is vertically concatenated with the *State_PendingJob* matrix, ultimately resulting in a matrix that captures the complete features of both the resources and the jobs across the multi-cloud data centers. To align these two concatenated matrices, we added a column of zeros to the matrix formed by concatenating *Latent VM encoding* and *State_PS*.

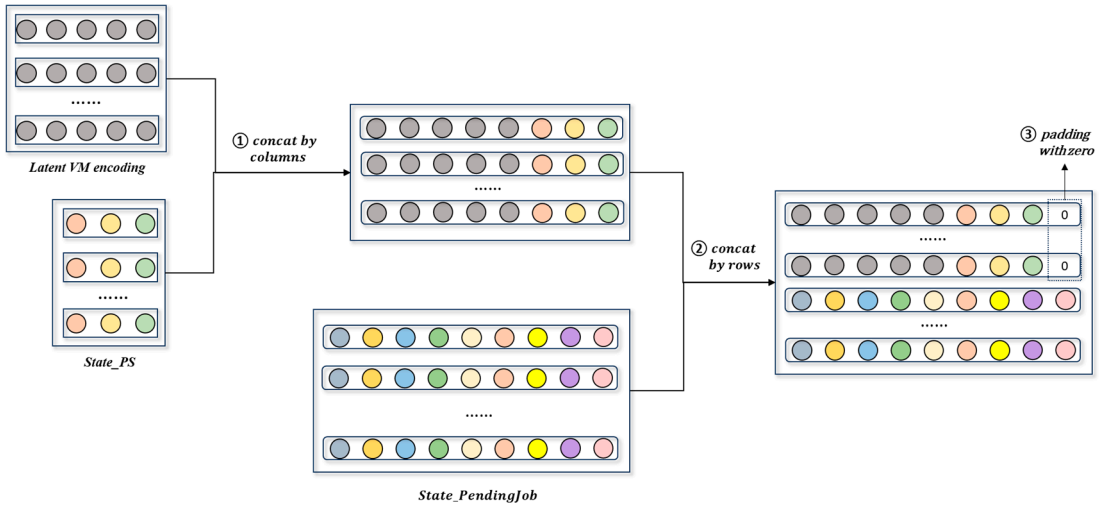


Figure 8. Diagram of Concat.

3. DC_Encoder

To more effectively extract and utilize the latent features of the overall state of multi-cloud data centers, the DC_Encoder component is designed as a convolutional neural network composed of two convolutional layers and one spatial pyramid pooling (SPP) layer, as shown in Figure 9.

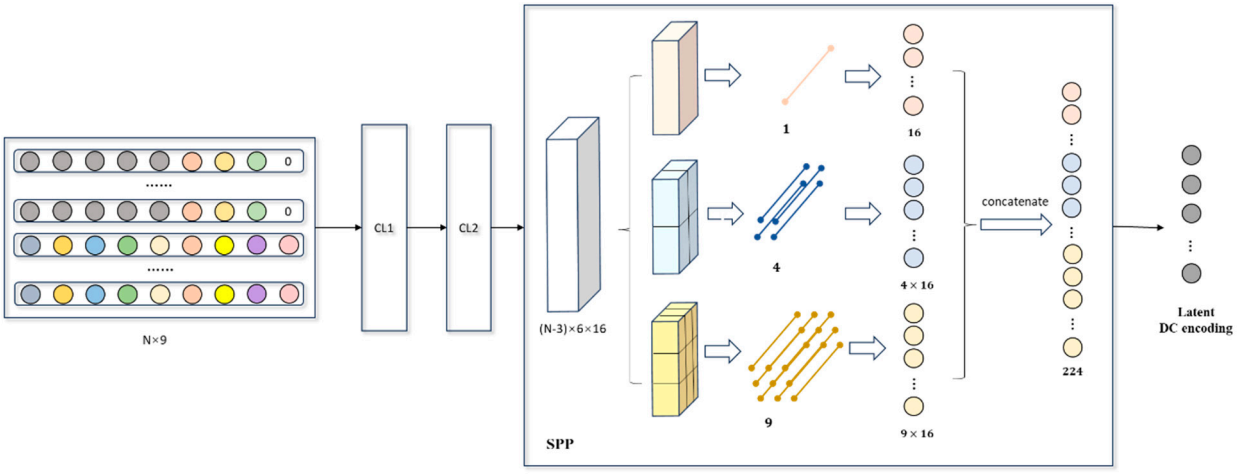


Figure 9. The architecture of the DC_Encoder.

Due to the varying number of pending jobs at different scheduling points, the number of rows in the overall state feature matrix of the multi-cloud data centers is uncertain. This matrix is first input into the two convolutional layers to extract higher-level feature information. Specifically, the first convolutional layer uses a 2×2 convolution kernel with a stride of 1 and 16 output channels. The second convolutional layer uses a 3×3 convolution kernel, also with a stride of 1 and 16 output channels. ReLU activation functions are applied after each convolutional layer.

Given that the number of rows in the overall state feature matrix is not fixed, while the fully connected layer in the subsequent Act_Dec component requires a fixed input dimension, the DC_Encoder introduces a spatial pyramid pooling (SPP) layer after the two convolutional layers. The SPP ensures that a fixed-dimensional feature vector is output, regardless of the input scale, by applying multiple pooling operations with different sizes. In our design, three adaptive pooling kernels are used, with sizes of 1×1, 2×2, and 3×3,

to pool the feature matrix. The pooled results are concatenated to produce a fixed-size output vector, referred to as *Latent DC encoding*.

4. Req_Predictor

Jobs in CDP applications exhibit the characteristics of random, continuous arrivals to the multi-cloud environment. Accurate prediction of the resource requirements and data processing scale for jobs arriving in the near future can provide more comprehensive information to aid scheduling decisions in the reinforcement learning model. To enable effective prediction, we employed Gated Recurrent Units (GRU) to construct the Req_Predictor component.

The architecture of the Req_Predictor component is illustrated in Figure 10. It takes *State_Hisreq* matrix as input and uses a two-layer GRU and two fully connected layers to extract the temporal features of jobs' resource requirements and data processing scales. Specifically, the component makes predictions based on the historical job information from the past 10 time units (each time unit being 30 s). Each layer of the two-layer GRU contains 32 neurons, and through the progressive processing of the two neural network layers, it effectively captures the long-term dependencies in the sequence data. The output dimension of the two-layer GRU is (10, 64), representing a 64-dimensional feature vector for 10 time steps. After flattening the output, a 640-dimensional one-dimensional vector is obtained.

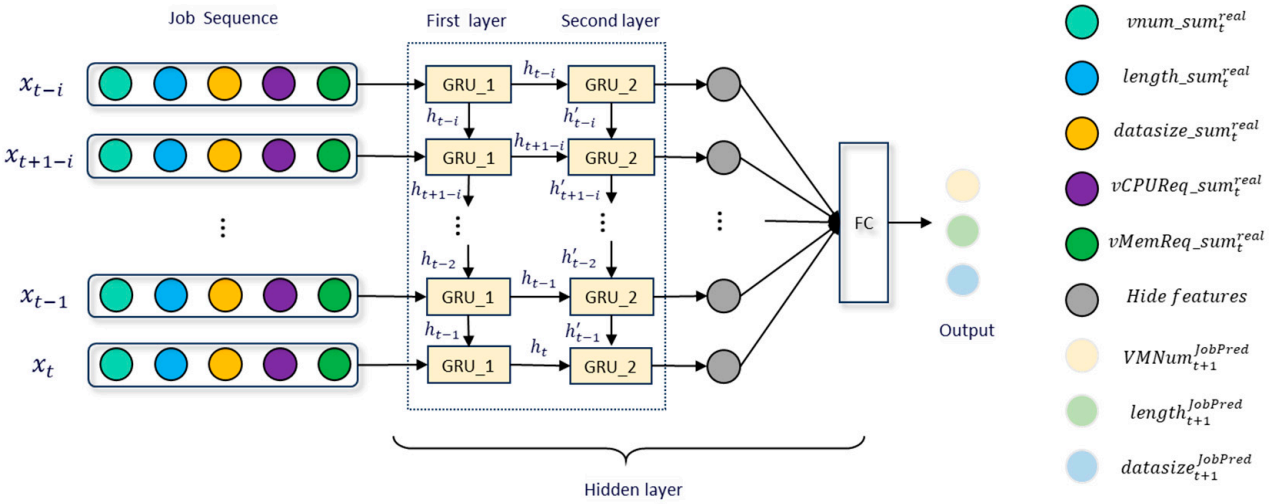


Figure 10. The architecture of Req_Predictor.

Subsequently, the model employs two fully connected layers for feature extraction and transformation. The first fully connected layer has 64 neurons, while the second contains 3 neurons. The output of the final layer provides the predicted job state for the next time unit, denoted as *Predicted Req*. These predicted values specifically include the total number of required VM instances, the incoming jobs' total computation length, and their total data processing scale, represented by $VMNum_{t+1}^{JobPred}$, $length_{t+1}^{JobPred}$, $datasize_{t+1}^{JobPred}$.

5. Act_Dec

As shown in Figure 11, the target scheduled job's state vector—extracted from the State_PendingJob matrix—is concatenated with *Latent DC encoding* and the *Predicted Req* to form the input to the Act_Dec component. This input encapsulates both the recent resource supply–demand status of the data centers and the explicit resource requirements and computational features of the job. The concatenated vector is then processed by a fully connected layer that maps it to a k-dimensional vector, where k is the number of actions in the action space. For the critic network, this k-dimensional vector is used directly as

the final output representing the value of each action. For the actor network, the vector is passed through a softmax function to produce a probability distribution over the actions, from which a valid action is selected.

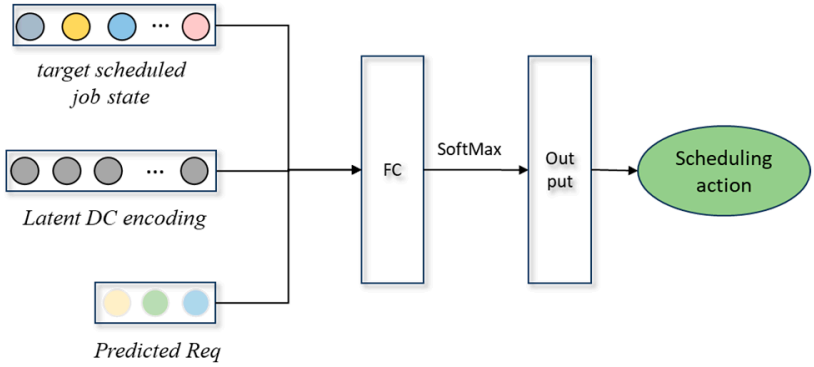


Figure 11. The architeture of Act_Dec.

5.2.2. Training of Actor and Critic Network

Among all the above components, the VM_Autoencoder and Req_Predictor are trained independently in an offline manner. The trained encoder and predictor are then integrated into the actor and critic networks. Specifically, the Req_Predictor is trained using the job submission sequence data from CDP applications in a multi-cloud environment, while the VM_autoencoder is trained on random samples of VM deployments and physical resource allocations in multi-cloud data centers. For the DC_Encoder and Act_Dec components, end-to-end training is employed.

When a large number of CDP applications are deployed in a multi-cloud environment, and as the types of physical and VMs in the data center increase, the state and action space for training the DC_Encoder and Act_Dec components becomes vast, significantly slowing model convergence. Additionally, the lag in obtaining new state and reward information after each job scheduling further hinders convergence. To address these challenges, we pretrained the DC_Encoder and Act_Dec components offline. During the pre-training stage, we used a genetic algorithm to generate high-quality decisions. Specifically, a collection of CDP applications is randomly generated, and preprocessing and aggregate analysis jobs are submitted at given intervals. Each job submission triggers the genetic algorithm, which selects an optimal resource allocation action for the job at the head of the queue from the action space. The fitness function of the genetic algorithm is based on the cumulative cost and expected SLA violation rate of the application. The algorithm uses the roulette method to select parent individuals for reproduction and adaptively adjusts the crossover and mutation rates based on the population's fitness distribution and evolutionary history. At each scheduling round, we calculated the reward based on the defined reward function and recorded the pre-scheduling state, the chosen resource allocation action, the updated post-scheduling state, and the reward as pretraining sample data. For the actor network pretraining, we used the mean squared error as the loss function, while for the critic network, the temporal difference error was used.

6. Performance Evaluation

We conducted both simulation and real-world experiments to evaluate the effectiveness of the proposed scheduling solution for CDP applications. In the simulation experiments, we thoroughly evaluated the performance of the CDP-EM model and the CDP-JS strategy within a simulated multi-cloud environment consisting of three distributed data centers. Subsequently, we assessed the scalability of the proposed solution by simulating multi-cloud environments with three, six, and nine data centers, respectively. In the

real-world experiment, we established an actual multi-cloud environment with three data centers located in Beijing and Nanjing, China, and employed representative benchmark workloads to build the CDP applications, demonstrating the superiority of our proposed solution.

6.1. Experimental Settings

In this section, we describe the detailed settings of the simulated multi-cloud environment with three data centers. The simulated environments for performance scalability evaluation and the real-world environment will be discussed in Sections 6.5.4 and 6.7, respectively.

We conducted simulation experiments to evaluate the effectiveness of the proposed CDP-EM model and CDP-JS strategy. The CDP application workloads in our study are generated using the cluster-trace-v2018 trace data, which are widely used in cloud computing research [44–46]. Released in 2018 by Alibaba, the largest internet company in China, cluster-trace-v2018 records detailed runtime information for over 2,100,000 application workloads from one of Alibaba's production cloud platforms with 4096 servers over eight days. The application workloads recorded in cluster-trace-v2018 generally consist of multiple jobs with complex DAG-style dependency structures. Furthermore, cluster-trace-v2018 captures detailed job-level runtime information, including the duration, resource requirements, resource usage, submission time, and more [47]. From the records of the first two days, we selected workloads with a two-layer structure of preprocessing and aggregate analysis, each containing more than four preprocessing jobs, resulting in 6273 simulated CDP applications with approximately 57,340 preprocessing jobs.

The arrival of simulated CDP applications follows a Poisson distribution with a default arrival rate of 0.1. The total data size processed by an application is obtained by multiplying the memory usage and execution duration of each preprocessing job within the application and then summing the products of all these jobs. The preprocessing jobs in an application are triggered when the accumulated processed data reach their corresponding data processing size recorded in the AT2018 trace. The resource requirement of a preprocessing or aggregate analysis job is determined based on the average CPU and memory usage, as well as the total number of parallel tasks. The computational load of a job is estimated by multiplying the floating-point operations per second (FLOPS) of its allocated CPU resources by its execution duration. An application's deadline is set as the sum of α times its turnaround time records in the trace and the application's submission time, where α is a random number in the range of [0.8, 1.5], simulating varying execution urgency requirements of the CDP applications.

We simulated a multi-cloud environment comprising three data centers. Each data center offered four VM types, with configurations and pricing based on Microsoft Azure, Amazon EC2, and the Google Cloud Platform. The VMs are deployed as on-demand instances. Detailed VM configurations and default pricing are listed in Table 1, physical server configurations in each data center are provided in Table 2, and inter-datacenter bandwidths are shown in Table 3.

The simulation experiments run on a server configured with an Intel Core i7-9700 processor, 32 GB RAM, 1 TB hard disk, and Windows OS. Python v3.10 was chosen as the main programming language. The trace data from 8 consecutive hours from 6 a.m. to 2 p.m. on the first day were used for the VM_Autoencoder and Req_Predictor model offline training and PPO network pretraining. The remaining trace data were used as test data.

Table 1. VM settings in the simulated multi-cloud environment.

Data Center	VM Type	vCPU	Mem (GB)	Computing Power (GFLOPS)	Per Hour ($)
Center 1	a1.small	2	8.60	35.2	0.0832
	a1.middle	4	17.18	70.4	0.166
	a1.large	8	34.36	140.8	0.333
	a1.xlarge	16	68.72	281.6	0.666
Center 2	b1.small	2	8.59	36.8	0.0816
	b1.middle	4	17.18	73.6	0.1632
	b1.large	8	34.36	147.2	0.3264
	b1.xlarge	16	64.00	294.4	0.6528
Center 3	c1.small	2	1.80	40	0.0709
	c1.middle	4	3.60	80	0.1418
	c1.large	8	7.21	160	0.2836
	c1.xlarge	16	14.42	320	0.5672

Table 2. Physical server configurations in the simulated multi-cloud environment.

Data Center	Cluster Type	Cluster Number	CPU Core Number	Core Computing Power (GFLOPS)	Mem (GB)
Center 1	Cluster1	10	50	35.2	161
Center 2	Cluster2	10	50	36.8	161
Center 3	Cluster3	10	50	40	40

Table 3. Bandwidth settings between simulated cloud data centers.

Source Data Center	Destination Data Center	Bandwidth
Center 1	Center 2	800 Mbps
Center 2	Center 3	600 Mbps
Center 1	Center 3	1 Gbps

6.2. Performance Metrics

We adopted two metrics to evaluate the performance of CDP-EM and CDP-JS: the *Cost* and *SLA_violation_rate*. The *Cost* metric represents the average resource rental cost for all simulated CDP application runs, while the *SLA_violation_rate* metric denotes the percentage of simulated CDP applications whose completion time exceeds the specified deadlines. The formulas for these two metrics are as follows:

$$Cost = \frac{1}{m}\sum_{i=1}^{m} cost_i \tag{27}$$

$$SLA\_violation\_rate = \frac{\sum_{i=1}^{m} SLA_i}{m} \tag{28}$$

where m is the number of all simulated CDP applications.

6.3. Baseline Methods

In our experiments, the CDP-EM and CDP-JS proposed in this paper are compared with the mainstream application execution models and scheduling algorithms in the current cloud environment. The baseline execution models include the following:

- One-off Execution (OE): Source data are continuously generated and accumulated until the acquisition deadline. Once reached, all the accumulated data are consolidated into a single job, which is then submitted to the cloud for preprocessing and aggregate analysis in one go.
- Anonymous Intermittent Execution (AIE): The accumulated source data are preprocessed in batches, with each batch corresponding to a preprocessing job. Once all preprocessing jobs are completed, the aggregate analysis job is submitted. All job submissions are manually triggered by the user and executed independently without reflecting any application ownership relationships.

In our experiment, we set the trigger event for the preprocessing job submissions in both AIE and CDP-EM to be the accumulation of data reaching a specified size. The difference is that the CDP-EM can automatically submit jobs via the agent based on the user-specified trigger events and is capable of recognizing the application ownership of the jobs.

The baseline job scheduling strategies are as follows:

- Random [26]: Jobs are selected from the job queue and assigned with VM resources in a random manner.
- HEFT [13]: Prioritizes jobs for execution based on their estimated job completion times, taking into account both the computation and communication costs in a heterogeneous computing environment.
- PSO-MC [19]: A hybrid scheduling algorithm that integrates particle swarm optimization (PSO) with membrane computing (MC). It leverages job and resource states to determine the optimal scheduling solution, with the primary objective of minimizing the completion time and cost.
- DB-ACO [24]: An enhanced ant colony optimization (ACO) algorithm that is tailored for workflow scheduling in cloud environments. It incorporates deadline and budget constraints to ensure that the scheduling solution meets specific time and cost requirements.
- HCDRL [35]: A deep reinforcement learning-based cloud task scheduling strategy for multiple workflows. This strategy uses the continuity of task execution within workflows as a constraint, aiming at the performance, cost, and fairness of workflow tasks. The D3QN algorithm is employed to find the optimal solution. We have retained the performance and cost objectives of the tasks in this strategy.
- ATSIA3C [37]: A deep reinforcement learning-based cloud task scheduling strategy achieves a joint reduction in makespan and energy consumption through subtask granularity optimization. The A3C algorithm is employed to find the optimal solution. We have replaced the PPO algorithm used in our CDP-JS strategy with the A3C algorithm while retaining our scheduling strategy.

To evaluate the overall performance of the proposed complete solution consisting of the CDP-EM and CDP-JS, we combined baseline application execution models and job scheduling strategies to form ten baseline solutions for CDP applications in a multi-cloud environment. In addition, we also compare the CDP-EM and CDP-JS with their corresponding baseline methods, respectively.

6.4. Hyperparameter Settings

The detailed hyperparameter settings of the actor and critic networks in the PPO algorithm employed in CDP-JS are listed in Table 4. The actor and critic networks share the same architecture. The detailed hyperparameter settings for the actor and critic networks in the PPO algorithm used in CDP-JS are listed in Table 4. The actor and critic networks share the same architecture. Among them, the VM_Autoencoder and Req_predictor components are trained separately in an offline manner. The trained models are then integrated with the DC_encoder and Act_Dec components for end-to-end training.

Table 4. Hyperparameter settings in the proposed PPO algorithm.

Algorithm	Hyperparameters	Parameter Values
PPO Network (DC_Encoder+Act_Dec)	Discount Factor	0.95
	Actor Learning Rate	0.0001
	Critic Learning Rate	0.0002
	GAE Lambda	0.95
	Clip Parameter	0.2
	Batch Size	128
VM_Autoencoder	Learning Rate	0.001
	Kernel Regularization	L2
	Internal layer Activation Function	ReLu
	Output layer Activation Function	Sigmoid
	Optimizer	Adam
	Epochs	500
	Batch Size	32
Req_Predictor	Learning Rate	0.001
	Input Dimension	5
	Hidden Units	32
	Layers	2
	Epochs	500
	Batch Size	32

We set the clipping parameter (ε) of the PPO algorithm to 0.2 based on preliminary experiments and a review of the literature. The seminal paper on the PPO algorithm by Schulman et al. (2017), along with subsequent studies (e.g., OpenAI Baselines), widely adopts and validates $\varepsilon = 0.2$ as the default value, as it effectively balances convergence speed and stability in most scenarios [42]. Moreover, in our preliminary experiments, we compared the effects of ε values of 0.1, 0.2, and 0.3. The results indicated that with $\varepsilon = 0.1$, the policy updates were overly conservative, leading to slow convergence, while $\varepsilon = 0.3$ resulted in larger updates and significant fluctuations in rewards. In contrast, $\varepsilon = 0.2$ offered the best trade-off between convergence speed and stability. For consistency and reproducibility, we kept the ε value fixed throughout the subsequent training process. Additionally, we employed advantage function normalization to enhance the training stability of the employed PPO algorithm [48].

6.5. Simulation Experiments

We first evaluated the overall performance of the proposed complete solution consisting of the CDP-EM and CDP-JS and then validated the effectiveness of the CDP-EM and CDP-JS, respectively. Finally, we present the ablation study of the proposed PPO algorithm used in CDP-JS.

6.5.1. Overall Performance Evaluation

As described in Section 6.3, we compare our proposed solution, i.e., CDP-EM+CDP-JS, with ten solutions combined with the baseline application execution models and job scheduling strategies. The simulated multi-cloud data centers are set as Tables 1–3, and the CDP application arrival is set as default. The performance results are present in Figure 12. In the bar chart of Figure 12, different colors are used to represent various baseline application execution models. The horizontal axis lists different job scheduling strategies. The combination of these two elements indicates that each bar represents a different baseline solution.

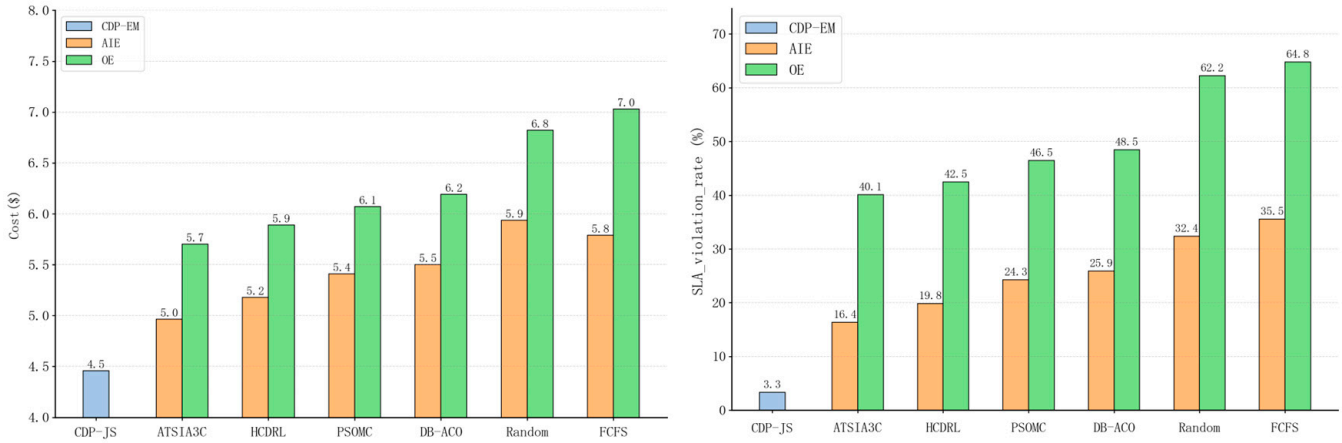


Figure 12. Performance comparison of CDP-EM+CDP-JS to baseline solutions: (**a**) cost and (**b**) SLA_violation_rate.

Compared to all baseline solutions, our proposed CDP-EM+CDP-JS solution achieves the best performance. The application SLA violation rate is reduced by an average of 34.8%, with a maximum reduction of 61.5%, while the resource cost is reduced by an average of 23.4%, with a maximum reduction of 36.6%. Overall, our solution exhibits a greater performance advantage compared to the baselines based on the OE model, reducing the application SLA violation rate by an average of 47.4% and resource cost by an average of 28.6%. Compared to the baseline solutions, which use the advanced scheduling strategy (ATSIA3C), CDP-EM+CDP-JS can still reduce the SLA violation rate and resource cost by an average of 24.9% and 16%, respectively. This superiority arises because the OE model only performs preprocessing and aggregation analysis operations after all data of a CDP application have been generated. Consequently, it fails to effectively leverage the idle and low-cost cloud computing resources available in the multi-cloud environment throughout the long data generation period, thereby reducing application efficiency and increasing the cost.

Regarding resource cost performance, our proposed solution achieves a more significant improvement in the application SLA violation rate. This is primarily due to the experimental setting, where a fixed pricing model for resources is adopted, meaning that the price of VM resources remains unchanged during application execution. This limits the opportunity for CDP jobs, especially preprocessing jobs, to take advantage of price variations caused by changes in supply and demand over extended periods. On the other hand, our solution explicitly accounts for application completion deadlines and execution progress, which contributes to its superior performance in reducing the SLA violation rate.

6.5.2. Performance Evaluation of CDP-EM

In this section, we adopt CDP-JS as the cloud job scheduling strategy and perform a comparative evaluation of CDP-EM against the baseline application execution models across various multi-cloud scenarios. Based on the settings in Tables 1–3, we vary the application arrival intensity, the number of physical servers, and VM resource pricing in the three data centers to simulate different multi-cloud scenarios. Specifically, the application arrival intensities are set as 0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175, and 0.2. The numbers of physical servers in each data center are configured as 5, 7, 10, 13, and 15. The VM resource prices are set based on the baseline prices in Table 1, with variation multipliers ranging from [0.4, 0.6], [0.6, 0.8], [0.8, 1], [1, 1.2], and [1.2, 1.4]. By varying the above factors, we simulated different resource supply and demand patterns in multi-cloud data centers. By adjusting the VM resource prices, we simulated different levels of the resource price variations between data centers.

As shown in Figure 13, CDP-EM outperforms the baseline application execution models across all scenarios. Compared to the AIE model, CDP-EM reduces the resource cost by an average of 3.55%, with a maximum reduction of 5.64%, and decreases the application SLA violation rate by an average of 9.01%, with a maximum reduction of 12.17%. In comparison to the OE model, the resource cost is reduced by an average of 17.24%, with a maximum reduction of 23.79%, while the application SLA violation rate drops by an average of 25.73%, with a maximum reduction of 31.81%. The performance of the OE model is the worst, as explained in the previous section. Our further statistical results prove that, with the OE model, the job waiting time is 35.23% longer compared to the CDP-EM model, and the cost of allocated VM resources is 21.64% higher. Compared to CDP-EM, the AIE model cannot effectively perceive the affiliation between applications and jobs, resulting in scheduling decisions often being made at the job level in isolation. This local optimization approach overlooks the long-term effects of overall application-level scheduling, reducing the likelihood of job co-location and continuous execution within an application, thereby increasing the data transfer time between jobs and the overall duration of the application.

As shown in the figure, with increased resource competition—higher application arrival intensity and fewer physical servers in data centers—the performance advantage of CDP-EM grows. Compared to the baseline model, the cost reduction increases from 6.2% to 15.1%, and the SLA violation rate decreases from 5.4% to 20.2%. CDP-EM effectively handles resource competition by intermittently submitting multiple preprocessing jobs, recognizing job-application affiliations, optimizing idle resources throughout the application lifecycle, and improving on-time completion. With larger resource price differences between data centers, CDP-EM's on-demand job submission mechanism efficiently utilizes low-cost resources, further reducing the application's resource costs.

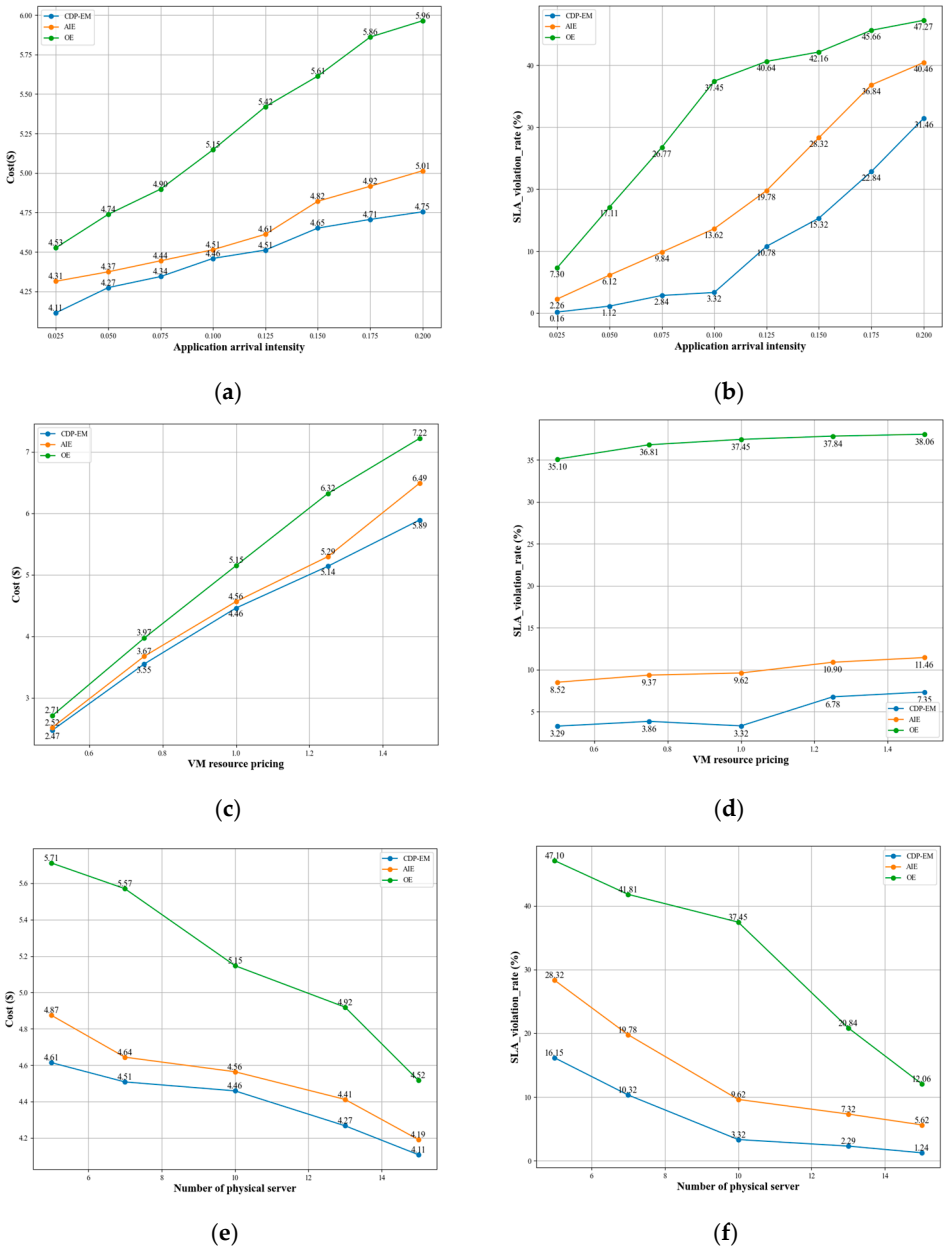


Figure 13. Performance comparison of CDP-EM to baseline job execution models: (**a**) Cost for different CDP arrival intensities; (**b**) SLA_violation_rate for different CDP arrival intensities; (**c**) cost for different VM resource pricings; (**d**) SLA_violation_rate for different VM resource pricings; (**e**) cost for different physical server numbers; (**f**) SLA_violation_rate for different physical server numbers.

6.5.3. Performance Comparison of CDP-JS

In this section, we fixed the application execution model as CDP-EM and compared the performance of CDP-JS against the five baseline cloud job scheduling strategies: HCDRL, PSOMC, DB-ACO, Random, and HEFT. We simulated the same multi-cloud scenarios as described in Section 6.5.2.

Figure 14 shows the performance comparison of the six scheduling strategies in different multi-cloud scenarios. The results indicate that CDP-JS consistently achieves the best performance across all scenarios. Specifically, compared to the five baseline strategies, CDP-JS reduces the resource cost by an average of 9.48%, 13.57%, 15.80%, 26.46%, and 25.89%, respectively, with the maximum reductions reaching 22.29%, 28.15%, 30.63%, 41.47%, and 42.65%. At the same time, in controlling the application SLA violation rate, CDP-JS also performs excellently, reducing the violation rate by an average of 5.92%,

10.76%, 11.60%, 24.30%, and 26.47%, respectively, with maximum reductions of 9.50%, 16.33%, 19.47%, 31.95%, and 35.97%.

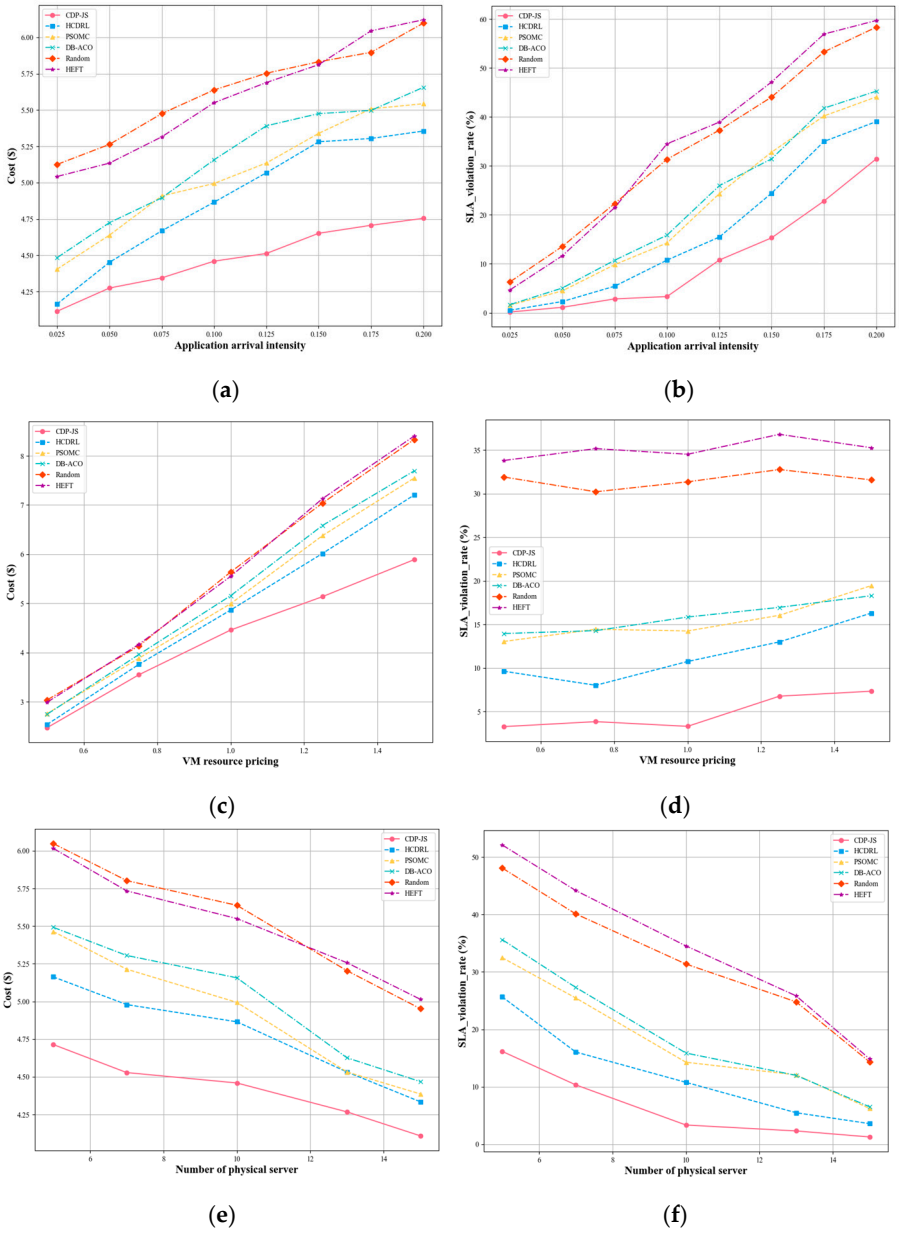


Figure 14. Performance comparison of CDP-JS to baseline strategies: (**a**) Cost for different CDP arrival intensities; (**b**) SLA_violation_rate for different CDP arrival intensities; (**c**) cost for different VM resource pricings; (**d**) SLA_violation_rate for different VM resource pricings; (**e**) cost for different physical server numbers; (**f**) SLA_violation_rate for different physical server numbers.

Random and HEFT are scheduling strategies that use simple, deterministic rules, resulting in the worst performance. This is because they lack the ability to dynamically adjust job execution priorities based on actual resource demands, leading to uneven resource allocation. Specifically, HEFT's focus on finish time alone can cause it to overlook the impact of long-running jobs on shorter ones, potentially increasing job waiting times. Additionally, neither of these strategies considers resource pricing, so changes in resource pricing have minimal impact on their scheduling strategies, leading to a linear increase in cost with resource pricing. In comparison, PSOMC and DB-ACO improve performance by using heuristic searching to explore the solution space for better job scheduling plans. However, when dealing with dynamically generated jobs for long durations, as in CDP

applications, these strategies struggle to coordinate job scheduling across multiple rounds. They are prone to becoming stuck in local optima and cannot dynamically adjust strategies to accommodate the complex and variable scheduling environment. The performance of the HCDRL algorithm is second only to CDP-JS. It uses a reinforcement learning model to dynamically adjust scheduling decisions based on real-time changes in the cloud environment and job status. However, HCDRL does not explicitly account for performance and cost at the workflow application level. Instead, it uses the execution continuity of jobs within a workflow (i.e., the interval between the start of consecutive jobs) as the reward factor. This reward definition is difficult to quantify in scenarios where CDP jobs are unpredictable and dynamically generated. Furthermore, the D3QN model it employs has weaker training stability and convergence speed compared to PPO, further reducing the scheduling efficiency of HCDRL.

Further observations of Figure 14 reveal that under relatively moderate resource competition, the performance of CDP-JS is optimal, but its advantage is not significant, as other strategies are also able to allocate reasonably configured and priced VM resources to jobs in a timely manner. However, as resource competition intensifies, the performance advantage of CDP-JS gradually increases, with the cost reduction rate rising from approximately 12.5% to 20.5% and the application SLA violation rate decreasing from 4.3% to 19.6%. This is due to CDP-JS's ability to swiftly learn historical job demand patterns and scheduling quality, allowing it to adapt to changes in the cloud environment and pending jobs, making optimized decisions that avoid potential SLA violation risks. Meanwhile, CDP-JS tries to schedule jobs to the lowest-cost VMs while meeting baseline resource requirements. Even in highly competitive environments, CDP-JS maintains relatively stable performance compared to other baseline strategies. Finally, it should be noted that when the application arrival intensity of CDP is extremely high, and data center resources are insufficient to support the concurrent execution of a large number of jobs, even with the optimal scheduling decision, CDP-JS performs relatively poorly, reflecting the real-world limitations of overloading data center resources.

6.5.4. Scalability Study

To evaluate the scalability of our proposed scheduling solution, we have expanded our existing multi-cloud environment by adding two simulated environments, clouds #2 and #3, increasing the total number of data centers to six and nine, respectively. In the expanded simulated multi-cloud environments, the new data centers first sequentially inherit the configurations of the original three data centers, as detailed in Table 2. Subsequently, we added the available virtual machine types and increased the arrival intensity of the CDP applications in the new simulated environments with the specific configurations provided in Table 5. In addition, the total time generated by the application is constant, so the total number of applications increases as the intensity of application arrivals increases. The detailed configurations of the new virtual machines are presented in Table 6. Notably, in Table 5, "New VM Location" indicates the number of data centers where the newly added virtual machine types can be deployed, while "VM Price Variation" refers to the multiplicative differences in pricing for the same type of virtual machine across different data centers, based on the base pricing outlined in Table 6. The applications' deadline setting is similar, as described in Section 6.1. In the network configurations from cloud #2 to cloud #3, each newly added group of three data centers adopts the bandwidth settings defined in Table 3. The bandwidth between each new data center and the existing ones is uniformly set at 800 Mbps.

Table 5. Settings in new simulated multi-cloud environments.

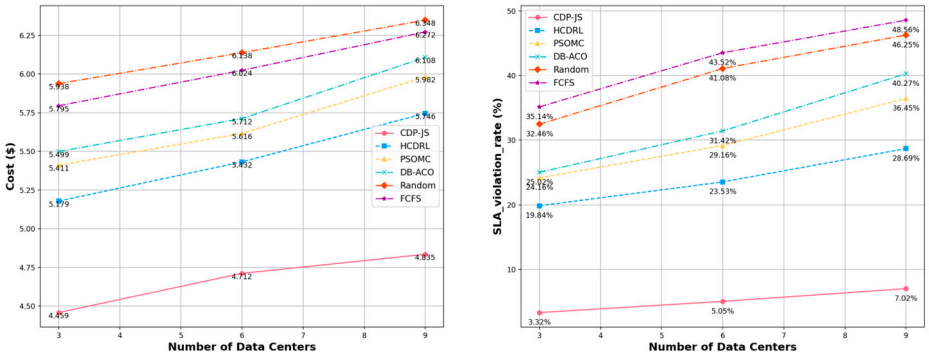
ID	No. of Data Centers	New VM Type	New VM Location	VM Price Variation	CDP Application Arrival Intensity
#2	6	d1. Micro, d1.xxlarge	3	1.2	0.2
#3	9	d1. Micro, d1.xxlarge e1. Micro, e1.xxlarge	4	1.4	0.3

Table 6. Configurations of new virtual machine types.

Data Center	VM Type	vCPU	Mem (GB)	Computing Power (GFLOPS)	Per Hour ($)
Center 4, 5, 6	d1. micro	1	4.30	17.6	0.0416
	d1.xxlarge	24	103.08	422.4	0.999
Center 7, 8, 9	e1. micro	1	4.30	18.4	0.0408
	e1.xxlarge	24	103.08	441.6	0.9792

Overall, in the simulated multi-cloud environments, as the number of data centers increases, there is a corresponding rise in the variety of virtual machine types, greater pricing variations among data centers, and an increase in the arrival intensity and total number of applications, thereby simulating larger-scale and more complex cloud environments.

We compared our proposed CDP-EM+CEP-JS scheduling solution with the baseline solutions that combine the AIE job execution model with baseline scheduling strategies. This comparison is based on preliminary evaluation results, which indicate that, among all baseline solutions, those utilizing the AIE execution model exhibit better performance. Figure 15 presents the experimental outcomes, demonstrating that as the scale of the multi-cloud environment expands, our proposed scheduling solution consistently maintains a performance advantage. In the three simulated multi-cloud environments, resource costs decreased by an average of 24.78%, with a minimum reduction of 17.06%; SLA violation rates decreased by an average of 30.86%, with a minimum reduction of 20.08%.



(**a**) (**b**)

Figure 15. Scalability comparisons CDP-EM+CDP-JS to baseline scheduling solutions: (**a**) cost and (**b**) SLA_violation_rate.

Further analysis across the three simulated environments reveals that the performance of the CDP-EM+CDP-JS scheduling solution does not deteriorate significantly with the expansion of the data center scale. Specifically, when the number of data centers increases from three to nine, the applications' resource costs and SLA violation rates increase by only 0.08 and 1.1 times, respectively. The stability in applications' resource costs underscores the

effectiveness of the CDP-EM job execution model, which distributes data preprocessing operations across different time periods to fully utilize idle and low-cost resources in the multi-cloud environment. At the same time, CDP-JS enhances scheduling performance scalability by dynamically optimizing decisions based on real-time cross-data-center resource states through feedback learning from historical scheduling decisions.

However, the increase in SLA violation rates surpasses that of the resource costs, primarily due to two factors: First, as the data center scale expands, we proportionally increase the application's arrival intensity, leading to a higher rate of job queuing that exceeds the growth in available resources. Second, the enlargement of the data center scale expands the state space encountered by the PPO algorithm within the CDP-JS strategy, thereby reducing the model's convergence speed.

To further analyze the scheduling decision-making capability of CDP-JS, we examined the stability and convergence speed of the online training of the employed PPO algorithm, as shown in Figure 16.

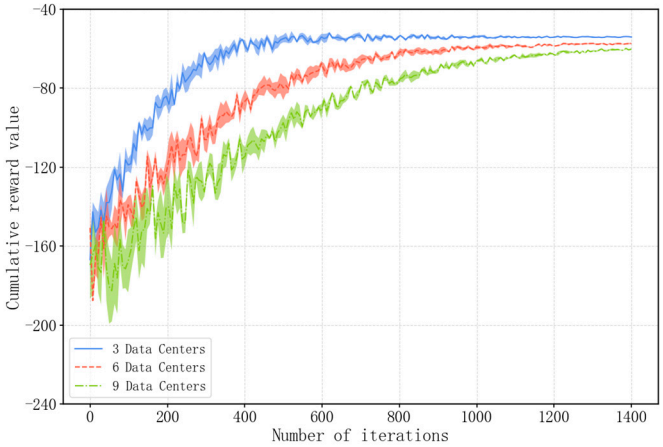


Figure 16. Convergence of the proposed PPO algorithm in different simulated multi-cloud scenarios.

The experimental results show that the PPO algorithm demonstrates strong convergence across multi-cloud environments of varying scales. As depicted in Figure 16, while increasing the scale of the multi-cloud environment extends the convergence time of the PPO algorithm (requiring 800, 1100, and 1400 iterations for three, six, and nine data centers, respectively), all learning curves eventually stabilize, confirming the robustness of the PPO algorithm in multi-cloud scaling scenarios. Further analysis reveals that as the number of data centers increases from three to nine, the final accumulated reward only decreases from -52 to -63, representing a relative reduction of 21.2%. This indicates that the PPO algorithm continues to perform well as the multi-cloud environment scales. Additionally, in all simulated environments, the accumulated reward consistently increases with each iteration, demonstrating that the model is effectively converging towards the optimal scheduling solution through an efficient policy update mechanism as the number of data centers grows.

6.5.5. Failure Resilience Study

In this section, we evaluate the performance of our CDP-EM+CDP-JS solution in a simulated multi-cloud environment with VM failures. The experiments were conducted using the simulation environment described in Section 6.5.1. We randomly terminated 10% and 20% of the deployed virtual machines, respectively, after they had been running for a period of time, with the runtime uniformly distributed between 10 and 50 s. Jobs running on the interrupted VMs were moved back to the pending job queue for rescheduling.

We compare the performance of our proposed solution with baseline scheduling solutions that integrate the AIE execution model with five baseline job scheduling strategies, as outlined in Section 6.3.

Table 7 presents the experimental results, demonstrating that in failure scenarios, the CDP-EM+CDP-JS solution consistently outperforms the baseline solutions. In scenarios with 10% interrupted VMs, resource costs are reduced by an average of 14.8%, with a minimum reduction of 8.26%; SLA violation rates are reduced by an average of 24.59%, with a minimum reduction of 10.23%. In scenarios with 20% interrupted VMs, resource costs are reduced by an average of 14.38%, with a minimum reduction of 7.81%; SLA violation rates are reduced by an average of 27.54%, with a minimum reduction of 12.01%. The performance advantage of the CDP-EM+CDP-JS solution is attributed to the fact that VM failures cause job interruptions, leading to rescheduling and intensifying resource competition. Our scheduling scheme optimizes decisions through feedback learning, selects jobs to schedule based on the application's execution progress, and matches optimal resources, thus achieving better performance and lower resource costs.

Table 7. Failure resilience comparison of CDP-EM+CDP-JS to baseline solutions.

		CDP-JS	**HCDRL**	**PSOMC**	**DB-ACO**	**Random**	**FCFS**
10% VM interruption	Cost ($)	4.918	5.362	5.477	5.681	6.216	6.246
	SLA_violation_rate (%)	4.60	14.83	19.39	21.19	43.89	46.63
20% VM interruption	Cost ($)	5.456	5.918	6.142	6.261	6.893	6.754
	SLA_violation_rate (%)	5.78	17.79	24.25	25.74	48.94	49.87

6.6. Ablation Study

In this section, we evaluate the performance contributions of the components employed in the actor and critic networks within the proposed PPO algorithm in our CDP-JS strategy. We conducted the ablation study in the simulated three-data-center cloud environment. In each experiment, we ablated a selected component and adopted an alternative approach. The models before and after ablation are referred to as Original and Ablation, respectively. The experiment settings are set as default (the same as the settings in the overall performance evaluations).

6.6.1. VM_Autoencoder

To assess the performance impact of the VM_Autoencoder component, we removed it from the proposed actor and critic networks and directly used the $State\_VM$ matrix as the $Latent\ VM\ encoding$ feature, which is concatenated with $State\_PS$ and $State\_PendingJob$ to represent the states of the multi-cloud data centers.

As shown in Table 8, with the ablation of VM_Autoencoder, the application SLA violation rate increased by 1.88%, and the average resource cost increased by approximately 2.64%. The results prove that, by encoding the VM state into a low-dimensional latent space, the VM_Autoencoder not only reduces computational complexity but also improves the quality of the features used in the actor and critic networks. This results in more efficient and accurate resource allocation and job scheduling decisions in multi-cloud data centers.

Table 8. Performance impacts of VM_Autoencoder.

	Original	**Ablation**
Cost ($)	4.359	4.442
SLA_violation_rate(%)	5.32	7.20

6.6.2. Req_Predictor

To evaluate the performance impact of the Req_Predictor component, we conducted two ablation experiments. In the first, we removed both the Req_Predictor component and its input, *State_Hisreq*, referred to as Ablation_Str+Input. In the second, we retained the *State_Hisreq* input and replaced the GRU-based Req_Predictor component with a two-layer fully connected network, referred to as Ablation_Str.

As shown in Table 9, without the State_Hisreq feature and the corresponding GRU-based Req_Predictor component, the scheduling performance deteriorates, with the average resource cost increasing by approximately 2.64% and the application SLA violation rate rising by 3.28%. This result demonstrates that using job history information to predict future resource demands in a multi-cloud environment can significantly improve scheduling performance. The evaluation of Ablation_Str validates that using the GRU effectively captures the temporal patterns in the historical resource demand variations of CDP jobs, thereby improving the accuracy of future resource demand predictions and better supporting job scheduling decisions.

Table 9. Performance impacts of Req_Predictor.

	Original	**Ablation_Str+Input**	**Ablation_Str**
Cost ($)	4.359	4.474	4.428
SLA_violation_rate (%)	5.32	8.60	8.2

6.6.3. PPO Algorithm

To assess the performance impact of the PPO algorithm employed in CDP-JS, we replace it with the D3QN algorithm, referred to as Ablation_D3QN. Both groups maintained identical experimental conditions, including the training dataset, state space definition, and reward function design.

As shown in Table 10, the experimental results demonstrate significant performance improvements: the average application cost was reduced by approximately 2.91% with the adoption of the PPO algorithm in CDP-JS, and the SLA violation rate was reduced by approximately 5.44%. The performance enhancement can be attributed to the PPO algorithm's introduction of a trust region constraint and clipped surrogate objective, which effectively controls the step size of policy updates. Compared to the ε-greedy exploration mechanism of D3QN, this constraint ensures stable policy iteration, avoiding the resource allocation oscillations caused by Q-value overestimation and thereby reducing the resource costs. Additionally, PPO's trajectory-based generalized advantage estimation strengthens the modeling capability of application and job dependencies, optimizing long-term returns and reducing the SLA violation rate.

Table 10. Performance impacts of the adoption of the PPO algorithm.

	Original	**Ablation_D3QN**
Cost ($)	4.359	4.486
SLA_violation_rate (%)	5.32	10.76

6.6.4. Pretraining of Proposed PPO Algorithm

We finally evaluated the impact of pretraining on our proposed PPO algorithm in CDP-JS, in which we have specifically designed actor and critic networks, both of which share the same architecture. The accumulated reward values across iterations are compared between PPO algorithms trained with and without pretraining.

As shown in Figure 17, the blue curve (pretraining) demonstrates a significantly faster convergence rate and lower variance in the early training stages compared to the red dashed curve (without pretraining). Specifically, at iteration 200, the pretrained PPO algorithm achieves an accumulated reward of approximately −80, while the non-pretrained algorithm remains around −240, experiencing high fluctuations. By iteration 400, the pretrained model stabilizes at −50, whereas the non-pretrained model still lags at −120. Eventually, both models converge to a similar performance level of approximately −40 around iteration 1000, but the pretrained model requires 50% fewer iterations to reach stability.

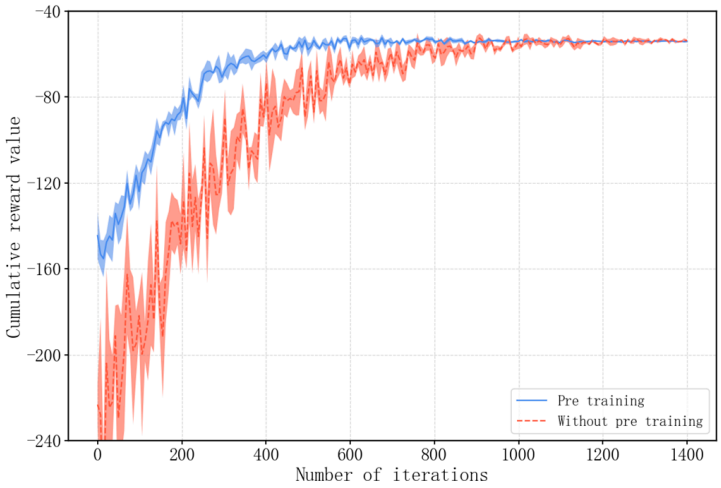


Figure 17. Performance impact of pretraining on the proposed PPO algorithm.

These results indicate that pretraining significantly enhances the learning efficiency of the customized actor–critic networks, enabling the PPO algorithm to achieve more stable and efficient policy updates. The shared architecture between the actor and critic networks benefits further from pretraining, as the learned feature representations contribute to both policy optimization (actor) and value estimation (critic). This experiment confirms that incorporating pretraining into our proposed PPO-based CDP job scheduling framework improves sample efficiency, accelerates convergence, and ultimately leads to more effective scheduling decision-making.

6.7. Real-World Experiments

In this section, we construct a real-world multi-cloud environment to validate the effectiveness of the CDP-EM+CDP-JS scheduling solution. The experiment was conducted across three data center clusters, located at Beijing Computing Center (BCC), the Institute of Computing Technology, the Chinese Academy of Sciences (ICT), and Nanjing Institute of InforSuperBahn (NISB), respectively. The BCC, ICT, and NISB data center clusters each contain five physical servers equipped with Intel Xeon Gold 5118, Intel Xeon Gold 6130, and Intel Xeon Gold 5120T processors, respectively. These processors feature 12, 16, and 14 CPU cores, and the servers are equipped with 96, 128, and 112 GB of memory, respectively.

Each data center implements dynamic virtual machine deployment via Kubernetes (K8S), with a maximum of five virtual machines deployed per physical server. The virtual machine configurations are referenced in Table 11.

The data centers are interconnected through BGP leased lines, and the network bandwidth configurations are shown in Table 12:

Table 11. VM settings in the real-world environment.

Data Center	VM Type	vCPU	Mem (GB)	Computing Power (GFLOPS)	Per Hour ($)
BCC	a1.small	2	8	36.8	0.0832
	a1.middle	4	16	73.6	0.166
	a1.large	8	32	147.2	0.333
	a1.xlarge	12	64	220.8	0.666
ICT	b1.small	2	8	33.6	0.0816
	b1.middle	4	16	67.2	0.1632
	b1.large	8	32	134.4	0.3264
	b1.xlarge	12	64	201.6	0.6528
NISB	c1.small	2	8	35.2	0.0709
	c1.middle	4	12	70.4	0.1418
	c1.large	8	32	140.8	0.2836
	c1.xlarge	12	64	211.2	0.5672

Table 12. Bandwidth settings between real-world data centers.

Source Data Center	Destination Data Center	Bandwidth
BCC	ICT	8 Gbps
BCC	NISB	4 Gbps
ICT	NISB	10 Gbps

Four typical CDP applications were designed for this experiment, covering various computing paradigms and data characteristics. The operations performed during the preprocessing and aggregation analysis phases, as well as the data sources processed, are shown in Table 13. The operation programs for both phases were derived from the BigDataBench benchmark suite [49].

Table 13. Application description.

Application Name	Preprocessing Stage	Aggregate Analysis Stage	Dataset
E-commerce Customer Behavior Analysis	Grep: Identifying Target User Group	K-Means: Customer Classification	E-Commerce Transaction Data [50]
Web Page Indexing	WordCount: Webpage Keyword Extraction	H-Index: Webpage Indexing	Sogu Data [51]
Movie Recommendation	Filter: Removing Invalid Records	CF: Collaborative Filtering Recommendation	MovieLen Data [52]
Community Discovery	Select Query: Selecting Key Attributes	Connected Component: Community Detection	Facebook Social Network Data [53]

The experiment constructs the CDP application workload set by randomly selecting from the four typical applications mentioned above, generating a total of 216 CDP applications. The data generation period spans 2 to 4 h. For each application, data records are randomly selected from the given dataset. Data generation follows a uniform distribution. The data volume generated every 15 min is uniformly distributed between 4 GB and 6 GB. Once the data reaches a specified size, preprocessing jobs are generated and submitted.

The cumulative size is uniformly distributed between 3 GB and 5 GB. After the data generation period ends, aggregate analysis jobs are automatically triggered. The deadline of an application is set by adding α times its execution time to its submission time on servers equipped with an Intel Xeon Gold 5120T (Intel Corporation, Santa Clara, CA, USA), where α ranges from 2 to 5. The workload generator is deployed at the Beijing Computing Center, with CDP applications arriving according to a Poisson distribution at an average arrival rate of 1.2 applications per minute ($\lambda = 0.02$).

Based on the experimental setup described above, we compared the performance of the CDP-EM+CDP-JS solution with the baseline scheduling solutions that integrate the AIE execution model with the five baseline job scheduling strategies described in Section 6.3.

As shown in Table 14, the proposed CDP-EM+CDP-JS solution achieves the best performance. Specifically, the SLA violation rate for CDP applications is reduced by an average of 27.2%, with a minimum reduction of 19.7%, while resource costs decrease by an average of 31.3%, with a minimum reduction of 23.1%. The results from the real-world experiments are consistent with the trends observed in the simulation experiments in Section 6.5.1, providing even stronger evidence. It is important to note that while the performance in the real-world environment is slightly lower than in the simulation environment, this is primarily due to the inherent complexity and variability of real-world conditions, such as network fluctuations, data transmission delays, and resource contention. However, the performance advantage of the CDP-EM+CDP-JS solution is more pronounced in the real-world environment, with a 6.5% greater reduction in resource costs compared to the simulation environment and a 3.2% larger reduction in SLA violation rates. These results highlight the applicability of our solution in real-world scenarios.

Table 14. Performance comparison of CDP-EM+CDP-JS to baselines in a real-world environment.

	CDP-JS	**HCDRL**	**PSOMC**	**DB-ACO**	**Random**	**FCFS**
Cost ($)	5.832	7.579	8.532	8.619	9.021	8.835
SLA_violation_rate (%)	4.58	24.32	27.68	28.42	38.32	40.32

7. Discussion

In multi-cloud environments, ensuring data security and integrity is crucial for the successful execution of cross-cloud job scheduling [54]. To address this, several security measures should be integrated into the CDP-EM+CDP-JS scheduling solution. First, during the intermidate data transferring between the preprocessing job and aggregate analysis job, data should be encrypted using strong encryption algorithms, along with proper key management, to prevent unauthorized access [55]. Secure communication protocols such as TLS/SSL are essential to protect data during transmission between different cloud data centers [56]. To maintain data integrity, the scheduling solution should implement cryptographic hashing and checksums, enabling the detection of any data corruption or unauthorized modifications during transfer or storage [57]. Additionally, redundancy and regular backups across multiple cloud data centers should be incorporated to ensure intermediate data availability and prevent loss during the long-term execution of CDP applications [58]. These measures will enhance data security and integrity, ensuring reliable job execution and data protection throughout the scheduling process.

Another discussion revolves around price fluctuations in the cloud computing market. On-demand instances and spot instances are the most widely used pricing models by cloud service providers [59]. On-demand instances maintain stable resource prices over a period of time, while spot instances dynamically adjust prices based on the supply and demand of data center resources. The CDP-EM+CDP-JS scheduling solution is based on

on-demand instances, assuming constant resource prices during decision-making. When adapting the proposed scheduling solution to scenarios with price fluctuations, mechanisms such as transfer learning can be employed to reduce the re-modeling cost of scheduling decisions and enable the scheduling model to adapt to the dynamic pricing of multi-cloud environments [60].

8. Conclusions

In this paper, we propose a novel cloud job execution model, CDP-EM, and a reinforcement learning-based job scheduling strategy, CDP-JS, forming a complete job scheduling solution for cumulative data processing (CDP) in multi-cloud environments. The proposed solution adapts to the unique characteristics of CDP applications where the processed data accumulate in a long-term and unpredictable manner by on-demand generations of preprocessing jobs, dynamically tracking application-level workload execution progress and employing feedback-based learning to optimize scheduling decisions. These measures ensure the high-performance, low-cost operation of CDP applications in multi-cloud environments. The performance evaluation results prove that the proposed multi-cloud job scheduling solution can reduce the SLA violation rate of CDP applications by an average of 34.8% and decrease resource costs by an average of 23.4%. In future work, we will extend the solution to support a resource pricing model based on Spot instances in cloud environments, improve our solution to accommodate scenarios involving unexpected faults and events and that concern data security and privacy issues during job scheduling.

Author Contributions: Conceptualization, Y.L. and G.X.; Data curation, G.X. and N.R.; Formal analysis, Y.L. and G.X.; Funding acquisition, Y.L.; Investigation, Y.L., G.X. and H.S.; Methodology, Y.L. and G.X.; Resources, G.X. and N.R.; Software, G.X. and Y.W.; Validation, Y.L. and G.X.; Visualization, G.X.; Writing—original draft, Y.L. and G.X.; Writing—review & editing, Y.L., G.X., H.S. and Y.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China; Grant Number: 62276011.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The dataset used for the experiment is publicly available at the following URL: https://github.com/alibaba/clusterdata/blob/v2018/cluster-trace-v2018/trace_2018.md (accessed on 5 October 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Shakor, M.Y.; Khaleel, M.I. Recent Advances in Big Medical Image Data Analysis Through Deep Learning and Cloud Computing. *Electronics* **2024**, *13*, 4860. [CrossRef]
2. Zhang, R.; Sun, Y.; Zhang, M. GPU-Based Genetic Programming for Faster Feature Extraction in Binary Image Classification. *IEEE Trans. Evol. Computat.* **2024**, *28*, 1590–1604. [CrossRef]
3. González-San-Martín, J.; Martinez, F.; Smith, R. A Comprehensive Review of Task Scheduling Problem in Cloud Computing: Recent Advances and Comparative Analysis. *New Horiz. Fuzzy Log. Neural Netw. Metaheuristics* **2024**, *1149*, 299–313. [CrossRef]
4. Shi, T.; Ma, H.; Chen, G.; Hartmann, S. Cost-Effective Web Application Replication and Deployment in Multi-Cloud Environment. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 1982–1995. [CrossRef]
5. Hou, H.; Agos Jawaddi, S.N.; Ismail, A. Energy Efficient Task Scheduling Based on Deep Reinforcement Learning in Cloud Environment: A Specialized Review. *Future Gener. Comput. Syst.* **2024**, *151*, 214–231. [CrossRef]
6. Cai, X.; Geng, S.; Wu, D.; Cai, J.; Chen, J. A Multicloud-Model-Based Many-Objective Intelligent Algorithm for Efficient Task Scheduling in Internet of Things. *IEEE Internet Things J.* **2021**, *8*, 9645–9653. [CrossRef]

7. Zhang, B.; Zeng, Z.; Shi, X.; Yang, J.; Veeravalli, B.; Li, K. A Novel Cooperative Resource Provisioning Strategy for Multi-Cloud Load Balancing. *J. Parallel Distrib. Comput.* **2021**, *152*, 98–107. [CrossRef]
8. Tekawade, A.; Banerjee, S. A Cost Effective Reliability Aware Scheduler for Task Graphs in Multi-Cloud System. In Proceedings of the 2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS), Bangalore, India, 3–8 January 2023; pp. 295–303. [CrossRef]
9. Tang, X. Reliability-Aware Cost-Efficient Scientific Workflows Scheduling Strategy on Multi-Cloud Systems. *IEEE Trans. Cloud Comput.* **2022**, *10*, 2909–2919. [CrossRef]
10. Hu, H.; Li, Z.; Hu, H.; Chen, J.; Ge, J.; Li, C.; Chang, V. Multi-Objective Scheduling for Scientific Workflow in Multicloud Environment. *J. Netw. Comput. Appl.* **2018**, *114*, 108–122. [CrossRef]
11. Xie, T.; Li, C.; Hao, N.; Luo, Y. Multi-Objective Optimization of Data Deployment and Scheduling Based on the Minimum Cost in Geo-Distributed Cloud. *Comput. Commun.* **2022**, *185*, 142–158. [CrossRef]
12. Jiang, F.; Ferriter, K.; Castillo, C. A Cloud-Agnostic Framework to Enable Cost-Aware Scheduling of Applications in a Multi-Cloud Environment. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–9. [CrossRef]
13. Khan, Z.A.; Aziz, I.A.; Osman, N.A.B.; Ullah, I. A Review on Task Scheduling Techniques in Cloud and Fog Computing: Taxonomy, Tools, Open Issues, Challenges, and Future Directions. *IEEE Access* **2023**, *11*, 143417–143445. [CrossRef]
14. Kanbar, A.B.; Faraj, K. Region Aware Dynamic Task Scheduling and Resource Virtualization for Load Balancing in IoT–Fog Multi-Cloud Environment. *Future Gener. Comput. Syst.* **2022**, *137*, 70–86. [CrossRef]
15. Zhou, X.; Zhang, G.; Sun, J.; Zhou, J.; Wei, T.; Hu, S. Minimizing Cost and Makespan for Workflow Scheduling in Cloud Using Fuzzy Dominance Sort Based HEFT. *Future Gener. Comput. Syst.* **2019**, *93*, 278–289. [CrossRef]
16. Alam, A.B.M.B.; Fadlullah, Z.M.; Choudhury, S. A Resource Allocation Model Based on Trust Evaluation in Multi-Cloud Environments. *IEEE Access* **2021**, *9*, 105577–105587. [CrossRef]
17. Liu, Z.; Xiang, T.; Lin, B.; Ye, X.; Wang, H.; Zhang, Y.; Chen, X. A Data Placement Strategy for Scientific Workflow in Hybrid Cloud. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 556–563. [CrossRef]
18. Meena, J.; Kumar, M.; Vardhan, M. Cost Effective Genetic Algorithm for Workflow Scheduling in Cloud Under Deadline Constraint. *IEEE Access* **2016**, *4*, 5065–5082. [CrossRef]
19. Li, K.; Jia, L.; Shi, X. Research on Cloud Computing Task Scheduling Based on PSOMC. *J. Web Eng.* **2022**, *21*, 1749–1766. [CrossRef]
20. Shi, T.; Ma, H.; Chen, G. A Genetic-Based Approach to Location-Aware Cloud Service Brokering in Multi-Cloud Environment. In Proceedings of the 2019 IEEE International Conference on Services Computing (SCC), Milan, Italy, 8–13 July 2019; pp. 146–153. [CrossRef]
21. Nabi, S.; Ahmad, M.; Ibrahim, M.; Hamam, H. AdPSO: Adaptive PSO-Based Task Scheduling Approach for Cloud Computing. *Sensors* **2022**, *22*, 920. [CrossRef]
22. Talha, A.; Malki, M.O.C. PPTS-PSO: A New Hybrid Scheduling Algorithm for Scientific Workflow in Cloud Environment. *Multimed. Tools Appl.* **2023**, *82*, 33015–33038. [CrossRef]
23. Kaur, A.; Singh, P.; Singh Batth, R.; Peng Lim, C. Deep-Q Learning-based Heterogeneous Earliest Finish Time Scheduling Algorithm for Scientific Workflows in Cloud. *Softw. Pract. Exp.* **2022**, *52*, 689–709. [CrossRef]
24. Tao, S.; Xia, Y.; Ye, L.; Yan, C.; Gao, R. DB-ACO: A Deadline-Budget Constrained Ant Colony Optimization for Workflow Scheduling in Clouds. *IEEE Trans. Automat. Sci. Eng.* **2024**, *21*, 1564–1579. [CrossRef]
25. Ullah, A.; Alomari, Z.; Alkhushayni, S.; Al-Zaleq, D.A.; Bany Taha, M.; Remmach, H. Improvement in task allocation for VM and reduction of Makespan in IaaS model for cloud computing. *Clust. Comput.* **2024**, *27*, 11407–11426. [CrossRef]
26. Wang, H.; Wang, H. Survey On Task Scheduling in Cloud Computing Environment. In Proceedings of the 2022 7th International Conference on Intelligent Informatics and Biomedical Science (ICIIBMS), Nara, Japan, 24–26 November 2022; pp. 286–291. [CrossRef]
27. Mao, H.; Schwarzkopf, M.; Venkatakrishnan, S.B.; Meng, Z.; Alizadeh, M. Learning Scheduling Algorithms for Data Processing Clusters. In Proceedings of the ACM Special Interest Group on Data Communication, Beijing, China, 19–23 August 2019; ACM: New York, NY, USA, 2019; pp. 270–288. [CrossRef]
28. Wang, P.; Xie, X.; Guo, X. Research on Resource Scheduling Algorithm for The Cloud. In Proceedings of the 2021 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS), Xi'an, China, 27–28 March 2021; pp. 732–735.
29. Guo, X. Multi-Objective Task Scheduling Optimization in Cloud Computing Based on Fuzzy Self-Defense Algorithm. *Alex. Eng. J.* **2021**, *60*, 5603–5609. [CrossRef]
30. Qin, Y.; Wang, H.; Yi, S.; Li, X.; Zhai, L. A Multi-Objective Reinforcement Learning Algorithm for Deadline Constrained Scientific Workflow Scheduling in Clouds. *Front. Comput. Sci.* **2021**, *15*, 155105. [CrossRef]
31. Zhao, F.A. Resource Scheduling Method Based on Deep Reinforcement Learning. *Comput. Sci. Appl.* **2021**, *11*, 2008–2018. [CrossRef]

32. Li, F.; Hu, B. DeepJS: Job Scheduling Based on Deep Reinforcement Learning in Cloud Data Center. In Proceedings of the Proceedings of the 2019 4th International Conference on Big Data and Computing—ICBDC 2019, Guangzhou, China, 10–12 May 2019; ACM Press: New York, NY, USA; pp. 48–53.
33. Mondal, S.S.; Sheoran, N.; Mitra, S. Scheduling of Time-Varying Workloads Using Reinforcement Learning. *AAAI* **2021**, *35*, 9000–9008. [CrossRef]
34. Ran, L.; Shi, X.; Shang, M. SLAs-Aware Online Task Scheduling Based on Deep Reinforcement Learning Method in Cloud Environment. In Proceedings of the IEEE 5th International Conference on Data Science and Systems (DSS), Zhangjiajie, China, 10–12 August 2019; pp. 1518–1525.
35. Chen, G.; Qi, J.; Sun, Y.; Hu, X.; Dong, Z.; Sun, Y. A Collaborative Scheduling Method for Cloud Computing Heterogeneous Workflows Based on Deep Reinforcement Learning. *Future Gener. Comput. Syst.* **2023**, *141*, 284–297. [CrossRef]
36. Zhang, S.; Zhao, Z.; Liu, C.; Qin, S. Data-intensive workflow scheduling strategy based on deep reinforcement learning in multi-clouds. *J. Cloud Comp.* **2023**, *12*, 125. [CrossRef]
37. Mangalampalli, S.; Karri, G.R.; Ratnamani, M.V.; Mohanty, S.N.; Jabr, B.A.; Ali, Y.A.; Ali, S.; Abdullaeva, B.S. Efficient deep reinforcement learning based task scheduler in multi cloud environment. *Sci. Rep.* **2024**, *14*, 21850. [CrossRef]
38. Abraham, O.L.; Ngadi, M.A.B.; Sharif, J.B.M.; Sidik, M.K.M. Multi-Objective Optimization Techniques in Cloud Task Scheduling: A Systematic Literature Review. *IEEE Access* **2025**, *13*, 12255–12291. [CrossRef]
39. Zhang, X.; Zhu, G.-Y. A Literature Review of Reinforcement Learning Methods Applied to Job-Shop Scheduling Problems. *Comput. Oper. Res.* **2025**, *175*, 106929. [CrossRef]
40. AWS. Amazon EC2 On-Demand Data Transfer Pricing [EB/OL]. Available online: https://aws.amazon.com/cn/ec2/pricing/on-demand/ (accessed on 5 March 2025).
41. Microsoft Azure. Bandwidth Pricing [EB/OL]. Available online: https://azure.microsoft.com/zh-cn/pricing/details/bandwidth/ (accessed on 5 March 2025).
42. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.I.; Abbeel, P. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
43. Ming, F.; Gong, W.; Wang, L.; Jin, Y. Constrained Multi-Objective Optimization with Deep Reinforcement Learning Assisted Operator Selection. *IEEE/CAA J. Autom. Sin.* **2024**, *11*, 919–931. [CrossRef]
44. Xu, M.; Song, C.; Wu, H.; Gill, S.S.; Ye, K.; Xu, C. esDNN: Deep Neural Network Based Multivariate Workload Prediction in Cloud Computing Environments. *ACM Trans. Internet Technol.* **2022**, *22*, 1–24. [CrossRef]
45. Liang, R.; Xie, X.; Zhai, Q.; Zhang, Q. Study on Container Cloud Load Prediction Based on Improved Stacking Integration Model. *Comput. Appl. Softw.* **2023**, *40*, 48–100.
46. Dang, W.; Zhou, B.; Wei, L.; Zhang, W.; Yang, Z.; Hu, S. TS-Bert: Time Series Anomaly Detection via Pre-Training Model Bert. In *Computational Science—ICCS 2021*; Lecture Notes in Computer Science; Springer International Publishing AG: Cham, Switzerland, 2021; Volume 12743, pp. 209–223. [CrossRef]
47. Alibaba Inc. Cluster Data Collected from Production Clusters in Alibaba for Cluster Management Research. 2018. Available online: https://github.com/alibaba/clusterdata (accessed on 15 February 2025).
48. Rio, A.D.; Jimenez, D.; Serrano, J. Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments. *IEEE Access* **2024**, *12*, 146795–146806.
49. BenchCouncil. BigDataBench [EB/OL]. Available online: https://www.benchcouncil.org/BigDataBench/index.html (accessed on 5 March 2025).
50. Ramos, G. E-commerce Business Transaction Sales Data. Kaggle. 2023. Available online: https://www.kaggle.com/datasets/gabrielramos87/an-online-shop-business (accessed on 15 February 2025).
51. Sogu Data Collection: Multimodal Social Media Analytics. 2023. Available online: https://www.selectdataset.com/dataset/966d4417a510d32a8423f2da627c342a (accessed on 15 February 2025).
52. Harper, F.M.; Konstan, J.A. The MovieLens datasets: History and context. *ACM Trans. Interact. Intell. Syst.* **2015**, *5*, 19. [CrossRef]
53. *Facebook Social Network Dataset*; Stanford University: Stanford, CA, USA, 2024. Available online: https://github.com/emreokcular/social-circle (accessed on 12 March 2024).
54. Kaur, A.; Dhiman, A.; Singh, M. Comprehensive Review: Security Challenges and Countermeasures for Big Data Security in Cloud Computing. In Proceedings of the 2023 7th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), Kolkata, India, 18–20 December 2023; pp. 1–6.
55. Sreekumari, P. Privacy-Preserving Keyword Search Schemes over Encrypted Cloud Data: An Extensive Analysis. In Proceedings of the 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), Omaha, NE, USA, 3–5 May 2018; pp. 114–120.
56. Soni, V.; Jain, V.; Santhoshkumar, G.P. Secure Communication Protocols for IoT-Enabled Smart Grids. In Proceedings of the 2024 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES), Chennai, India, 12–13 December 2024; pp. 1–6.

57. Vineela, A.; Kasiviswanath, N.; Bindu, C.S. Data Integrity Auditing Scheme for Preserving Security in Cloud Based Big Data. In Proceedings of the 2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 25–27 May 2022; pp. 609–613.
58. Emara, T.Z.; Huang, J.Z. Distributed Data Strategies to Support Large-Scale Data Analysis Across Geo-Distributed Data Centers. *IEEE Access* **2020**, *8*, 178526–178538. [CrossRef]
59. Choudhary, A.; Verma, P.K.; Rai, P. Comparative Study of Various Cloud Service Providers: A Review. In Proceedings of the 2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India, 8–9 December 2022; pp. 1–8.
60. Zhu, Z.; Lin, K.; Jain, A.K.; Zhou, J. Transfer Learning in Deep Reinforcement Learning: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *45*, 13344–13362. [CrossRef]