Journal of Network and Computer Applications 208 (2022) 103520

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Deep reinforcement learning-based algorithms selectors for the resource scheduling in hierarchical Cloud computing

Guangyao Zhou [a], Ruiming Wen [a], Wenhong Tian [a,*], Rajkumar Buyya [b,a]

[a] School of Information and Software Engineering, University of Electronic Science and Technology of China, China
[b] Cloud Computing and Distributed Systems Lab, School of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Keywords:
Hierarchical cloud computing
Algorithm selection
Subsystem
DL-based selector
DRL-based selector

ABSTRACT

Cloud computing environment is becoming increasingly complex due to its large-scale information growth and increasing heterogeneity of computing resources. Hierarchical Cloud computing dividing the system into multi-levels with multiple subsystems to support the adaptability to abundant requests from users has been widely applied and brings great challenges to resource scheduling. It is critical to find an effective way to address the complex scheduling problems in hierarchical Cloud computing, whose scenarios and optimization objectives often change with the types of subsystems. In this paper, we propose a scheduling framework to select the scheduling algorithms (SFSSA) for different scheduling scenarios considering no algorithm well suitable to all scenarios. To concretize SFSSA, we propose deep learning-based algorithms selectors (DLS) trained by labeled data and deep reinforcement learning-based algorithms selectors (DRLS) trained by feedback from dynamic scenarios to complete the algorithms selection regarding the scheduling algorithms as selectable tools. Then, we apply strategies including pre-trained model, long experience reply and joint training to improve the performance of DRLS. To enable the quantitative comparison of selectors, we introduce a weighted cost model for the trade-off between solution and complexity. Through multiple sets of experiments in hierarchical Cloud computing with multi subsystems for five types of scheduling problems and varying weights of cost, we demonstrate DLS and DRLS outperform baseline strategies. Compared with random selector, greedy selector, round-robin selector, single best selector, virtual best selector and single fast selector, DLS reduces the cost by 47.4%, 46.1%, 33.9%, 47.9%, 19.3%, 18.8% under stable parameter ranges, and DRLS reduces the cost by 41.1%, 40.6%, 11.7%, 42.3%, 11.5%, 12.5% in dynamic scenarios respectively. In experiments, we also validate DRLS has stronger adaptability than DLS in dynamic scheduling scenarios and DRLS using all of strategies achieves the best performance.

1. Introduction

The advent of Industry 4.0 and 5G Era is producing an increasing volume of data to Internet putting forward requirements for mighty software systems of network (Armbrust et al., 2010). Cloud computing, as a large-scale distributed system that provides flexible, reliable, dynamic and high coverage services, has shown significant advantages in meeting the demand of Internet and is playing an indispensable role in various professions including scientific research and engineering production (Adhikari et al., 2019). With its heterogeneous resources and policy of pay-as-you-use, Cloud computing can respond to different types of users' requests such as transmission requests, storage requests, computing requests, etc (Zhou et al., 2019; Kumar et al., 2019). Currently, Cloud computing has become the foundation for many business applications (Guo et al., 2021).

The increasing task requests and data transmission make the scale of Cloud computing system gradually expand based on deployment of hardware devices. However, the deployment capability lags behind the increase of Internet data, which presses exactly the management of resources in Cloud. In addition, inappropriate utilization ratio of resources will cause excessive energy consumption, high running time of tasks, and significant burden to systems so as to reduce the quality of service, reduce service life of components and increase CO_2 emission, which presents the necessity of requirement for efficient scheduling (Zhou et al., 2019; Kumar et al., 2019; Wan et al., 2020). Various factors containing time, resource state and environment, will affect the optimization objective and scenario, resulting in greatly high complexity of the solution process. Therefore, the research of resource

* Corresponding author.
E-mail addresses: guangyao_zhou@std.uestc.edu.cn (G. Zhou), ruimingwen0516@gmail.com (R. Wen), tian_wenhong@uestc.edu.cn (W. Tian), rbuyya@unimelb.edu.au (R. Buyya).

https://doi.org/10.1016/j.jnca.2022.103520
Received 1 June 2022; Received in revised form 22 July 2022; Accepted 22 September 2022
Available online 27 September 2022
1084-8045/© 2022 Elsevier Ltd. All rights reserved.

scheduling in Cloud computing has always been a hotspot and nodus in the era of big data, which also influences the position and development of Cloud computing in society.

One novel structure of hierarchical Cloud computing (including mobile edge computing) is to divide the Cloud system into multiple subsystems (Multi-Level Cloud System) to provide specific types of services for corresponding users, which reduces the computational complexity of resource management and achieve considerable performance (Lu et al., 2020; Li et al., 2020). Thus, research on multi Cloud environments is one of the tendencies to improve the application performance of Cloud (Hu et al., 2018). The potential user types are the basis of subsystem division and the types of services, which also cause the difference of equipment composition between different subsystems. This difference determines the various objectives of each subsystem and increases the difficulty of resource management. Resource management has been broadly studied and there are various scheduling algorithms such as heuristic, meta-heuristics and machine learning. Some algorithms, such as Ant colony algorithm (Sudarshan Chakravarthy et al., 2019), NSGA-II algorithm (Sofia and Ganeshkumar, 2018) and deep reinforcement learning (Guo et al., 2021; Li et al., 2020), have shown excellent performance in the existing research, whereas the scheduler based on single scheduling algorithm is far from meeting the demand of the actual operation environment, especially with multi subsystems as no algorithm can fit all scenarios of Cloud computing and perform better than other algorithms currently. Meta-heuristic, deep reinforcement learning and other algorithms, that can resolve the problem of variable optimization objectives and scenarios to a certain extent, require and consume extensive computing resources and time with the increasing complexity of the problem. And some algorithms with certain rapidity such as LPT, FCFS, RR, BFD and Greedy can only address the resource scheduling in simple scenarios with the low requirement as their solution is obviously worse than other complex algorithms (Guo et al., 2021; Mao et al., 2021).

The existing contradiction, that determines the upper limit of the profit, is the balance between optimization and complexity of the algorithm. Moreover, the increasing complexity of the algorithm will abate the realistic value to deploy and apply the algorithm as well as cause resource bottleneck and risk in the operation process of Cloud systems. Therefore, the improvement of one algorithm is inadequate to solve this contradiction. Some existing research on the resource scheduling in hierarchical Cloud computing applied DQN (Lu et al., 2020; Li et al., 2020), Joint DC (JCORA) algorithm (Nguyen et al., 2020), a penalized successive convex approximation (P-SCA)-based algorithm (Liu et al., 2020), and VCEPSO based on particle swarm optimization (Li, 2020) and achieved good performance in their researched problem. However, they mainly focused on the improvement of a single scheduling algorithm without considering the differences between subsystems and between their objectives. In reality, different subsystems in hierarchical Cloud computing may have different equipment compositions to provide different services for various users, which causes the change of optimization objectives concerned by resource scheduling of different subsystems. Therefore, it is considerable to dynamically select algorithms for variable scenarios. This prompts us to consider making full use of the existing algorithm to meet more complex scenes with various objectives in hierarchical Cloud computing with multi subsystems (HCCMS).

In the existing resource scheduling algorithms of Cloud computing and other distributed systems, a phenomenon from observation is that each optimization algorithm has its targeted factors and scenarios since its proposition. Considering another point of view that all the scheduling algorithms belong to human's wisdom and referring to research on algorithm selectors in other fields (Czako et al., 2021; Huerta et al., 2022), this paper proposes a scheduling framework to select the scheduling algorithms (SFSSA) by combining advantages of various scheduling algorithms. To concretize SFSSA, this paper also proposes deep learning-based algorithm selectors (DLS) and deep reinforcement learning-based algorithm selectors (DRLS) to deal with the resource scheduling problems of HCCMS with different optimization objectives and varying weights of cost. Although training DRL or DL to schedule resources directly will consume abundant computing capacities and time because the input and output spaces of resource scheduling are too large, DLS and DRLS only consume tiny complexities to select the scheduling algorithms. Finally, according to the types of scheduling problems, the weight of solution and complexity, as well as the number of tasks and resources, DLS or DRLS can select an algorithm with good performance from the algorithm pool to minimize the cost of HCCMS.

The main contributions of this paper can be summarized as:

(1) Scheduling framework to select the scheduling algorithms (SFSSA): this paper proposes a framework to select scheduling algorithm according to optimization objective and scenarios regarding the scheduling algorithms also as selectable resources, which can integrate the advantages of various scheduling algorithm, decompose the optimization objective and also effectively decompose the computational complexity of the optimization algorithm in complex resource management scenarios of HCCMS.
(2) DLS and DRLS: this paper utilizes deep learning (DL) and deep reinforcement learning (DRL) respectively to represent the decision-making process of selecting scheduling algorithm to construct DLS and DRLS. Using DL and DRL, the DLS and DRLS can effectively model multiple optimization scenarios and effectively train the strategy of scheduling algorithm selection, which adapts various scenarios of variable optimization objectives. Our proposed DLS and DRLS also explore a novel role for DL and DRL as scheduling algorithm selector of Cloud computing.
(3) Combination of various strategies of DRLS: to further improve the performance of DRLS, this paper applied three strategies in DRLS including pre-trained model, long experience replay and joint training, where each strategy optimizes the selection results of DRLs to a certain extent. The DRLS using all these three strategies performs best among all the DRL-based selectors.
(4) Multiple sets of experiments: this paper constructs scheduling algorithms pool and carries on multiple sets of experiments in complex scenarios of HCCMS to verify the superiority of proposed algorithm selector from various sights. In experiments, DLS and DRLS achieve better results in their own corresponding scenarios compared with baseline strategies including random selector (RS), greedy selector (GS), round-robin selector (RRS), single best selector (SBS), virtual best selector (VBS) and single fast selector (SFS).

The remainder of this paper is organized as follows. We review the related works in Section 2. System model and formulation of scheduling problems in complex scenarios of HCCMS are presented in Section 3. The scheduling framework to select the scheduling algorithms (SFSSA) as well as the DL-based and DRL-based algorithms selectors with multiple strategies are proposed in Section 4. The design and results of multiple sets of experiments from various sights are presented in Section 5. Finally, we conclude this paper in Section 6.

2. Related work

Frequently used approaches for resource scheduling of Cloud computing contain migration such as VMs migration (Sudarshan Chakravarthy et al., 2019), application migration (Duc et al., 2019), task migration (Miao et al., 2020), and workload migration (Fiandrino et al., 2017); Queuing model such as M/M/S (Ding et al., 2020) and M/G/1 (Li, 2009); multi-phase approach (Laili et al., 2020); as well as scheduling algorithm. The core of these approaches is still the scheduling algorithm including several categories according to the solution way: direct allocation algorithms, search algorithms, and machine learning algorithms.

Direct allocation algorithm, mostly heuristic algorithm, is one type of the commonly applied algorithms in realistic Cloud systems that benefit from its low computing complexity, light-weight, analyzability, ease of implementation and deployment. In the direct allocation algorithm, LPT (Croce and Scatamacchia, 2020), greedy (Mao et al., 2021), random, RR (Round-Robin) (Pradhan et al., 2016), FFD (Tian et al., 2018), FCFS (First Come First Serve) (Abhikriti and Sunita, 2017) are frequently utilized algorithms to gain an approximate solution of scheduling problems and have also been applied as baselines of comparison in recent literature. Z. Guan et al. applied Jacobi Best-response Algorithm and proposed a novel globally optimization algorithm based on a combination of the branch and bound framework to solve the resulting non-convex cooperative problem and minimize the costs of Mobile Cloud Computing (Guan and Melodia, 2017). Wenhong Tian et al. proposed a 2-approximation algorithm, LLIF, with theoretical analysis and proof to minimize the energy consumption of VMs scheduling (Tian et al., 2018). Z. Hong proposed FCFI + ACTI, a QoS-Aware Distributed Algorithm to address multi-hop computation-offloading problem in IoT-Edge-Cloud Computing (Hong et al., 2019).

Search algorithm, mainly meta-heuristic algorithm, is a local search or global search to get final scheme based on an initial allocation state, which can generally achieve better solution than direct allocation algorithm but will consume more complexity of time and space. This type of algorithm has adaptability to the varied objectives hence applies to multi-objective optimization problems. Xiao-Fang Liu et al. develop an ant colony system-based approach named OEMACS allocating VMs to reduce energy consumption of Cloud computing (Liu et al., 2018). M. Mahil et al. proposed PSO-ACS algorithm incorporating particle swarm optimization and Ant Colony System to optimize energy efficiency and SLA Violation of cloud data centers (Mahil and Jayasree, 2021). MOEAs (Laili et al., 2020) have performed superiority in multi-objective scheduling problems in Cloud computing. Ali Abdullah Hamed Al-Mahruqi et al. proposed HH-ECO, a Hybrid Heuristic Algorithm using chaotic based particle swarm optimization (C-PSO), to improve optimal makespan and energy conservation (Al-Mahruqi et al., 2021). Amir Iranmanesh et al. proposed DCHG-TS, a hybrid genetic algorithm based on load balancing routing, to optimize both cost and makespan for scientific workflow scheduling in Cloud computing (Iranmanesh and Naji, 2021).

Machine learning, mainly reinforcement learning and deep reinforcement learning usually occupying better optimization solutions in dynamic scheduling problems than search algorithms, adapts various scenarios apt to lifelong learning, but requires abundant data-based training with extensive computing complexity and possesses unpredictable execution results without analyzability, which may cause risk to large-scale Cloud systems in realistic. Combining neural network and NSGA-II algorithm, Goshgar Ismayilov et al. proposed the NN-DNSGA-II algorithm, a multi-objective evolutionary algorithm for dynamic workflow scheduling in Cloud computing (Ismayilov and Topcuoglu, 2020). Based on M/M/S queuing model and Q-learning, Ding Ding et al. proposed QEEC, a two phases framework, to enhance the energy efficiency of Cloud computing (Ding et al., 2020). However, the state space is high dimensional and continuous resulting in massive computing complexity to train Q-table in scheduling problems. K. Lolos et al. proposed MDP_DT, an adaptive Reinforce Learning with three strategies that Chain Split, Reset Split and Two-phase Split to reduce the size of state space (Lolos et al., 2017). One other strategy to express the continuous state space in reinforcement learning is the deep neural network. DRL, integrating reinforcement learning and deep learning, such as modified DRL algorithm (Karthiban and Raj, 2020), DQTS (Tong et al., 2020), Deep Q Network (DQN) (Dong et al., 2020), ADRL (Kardani-Moghaddam et al., 2021) and DeepRM-Plus (Guo et al., 2021), has been applied to solve scheduling problems in Cloud computing and performed the superiorities of DRL verified by experiments in their papers.

Table 1
Notations and descriptions.

Notation	Description
q	Number of subsystems
k	Index of subsystem
S_k	The subsystem with index k
n_k	The number of server nodes in S_k
N_k	The set of server nodes in S_k
i	Index of server node
R_{ki}	The ith server node of S_k
T_k	The set of tasks allocated to subsystem S_k
m_k	The number of tasks in subsystem S_k
j	Index of tasks
T_{kj}	The jth task in subsystem S_k
TS_{ki}	Set of tasks in server node R_{ki}
KS_k	The set of TS_{ki} where $KS = \langle TS_{k1}, TS_{k2}, \ldots, TS_{kn_k} \rangle$
C_{ki}^{CPU}	The maximum capacity of CPU for node R_{ki}
C_{ki}^{DS}	The maximum capacity of DS for node R_{ki}
v_{kji}^{CPU}	The CPU capacity request of T_{kj} for R_{ki}, similarly v_{kji}^{DS}
L_{ki}^{CPU}	The occupied CPU capacity of R_{ki}, similarly L_{ki}^{DS}
ET_{kji}	The processing time of task T_{kj} when executed in R_{ki}
RT_{ki}	The running time of server node R_{ki}
MT_k	The makespan of the kth subsystem S_k
$U_k(t)$	The cost of the kth subsystem S_k at time slot $[t, t+\delta t)$
$\Omega(t)$	The cost of the whole system at time slot $[t, t+\delta t)$

Additionally, for hierarchical or multi Cloud computing: Haifeng Lu et al. proposed IDRQN to improve the energy consumption, load balancing, latency and average execution time for MEC with multi-area subsystems (Lu et al., 2020); Meng Li et al. applied DDQN to improve the system performance for blockchain-enabled M2M communications in EC with multi groups of M2M network (Li et al., 2020); Hu et al. proposed a MOS (multi-objective scheduling) algorithm based on the particle swarm optimization (PSO) to minimize workflow makespan and cost simultaneously of multi-cloud environment (Hu et al., 2018); other algorithms including Joint DC (JCORA) algorithm (Nguyen et al., 2020), P-SCA-based algorithm (Liu et al., 2020), and PSO-based VCEPSO (Li, 2020) have been proposed to address the resource scheduling problems of hierarchical Cloud computing.

Overall, the existing research mainly targets improving the performance of the scheduling algorithm itself. Differentiating from previous research, this paper focuses on the algorithms selection of HCCMS, regards scheduling algorithms as selectable resources and proposes a scheduling framework to select the scheduling algorithms (SFSSA). Additionally, this paper accords a novel role, i.e. algorithms selectors, to DL and DRL by proposing DL-based selector (DLS) and DRL-based selector (DRLS) for various scenarios.

3. System model and problem formulations

To assist the description of the system model and problem formulations, Table 1 gives some notations used in this paper.

3.1. System model of multi-level Cloud system

Cloud computing integrates the physical or virtual machines of various servers through high-speed Internet to form a resources pool. When a user submits a task request, the Cloud computing management center allocates the resources in the resource pool to the user according to the current operation status of the Cloud computing system and the attributes of the user's request. After allocation, Cloud computing management updates the status of the Cloud system and monitors the operation of the whole system in real-time. The traditional process of Cloud management without subsystem can be seen in Fig. 1.

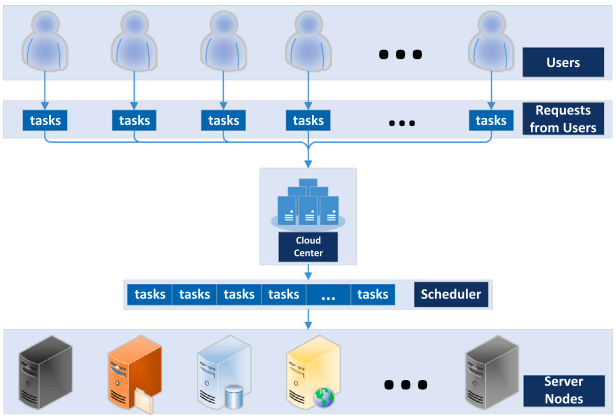


Fig. 1. The traditional process of Cloud resource management.

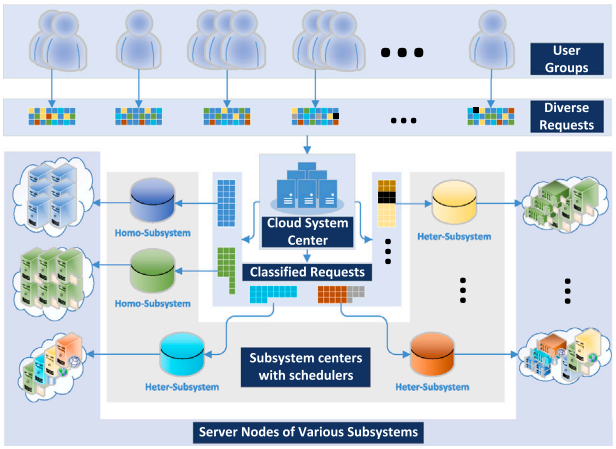


Fig. 2. The hierarchical Cloud computing with multi subsystems (HCCMS).

Currently, a single Cloud computing resource management center can no longer meet the rapidly growing number of task requests from all over the world. Moreover, a single resource management center is not robust with a low ability to anti risk. During actual operation, multi-point control is usually adopted to manage the Cloud system i.e. according to the attributes of Cloud computing server nodes and the classification of target users, the large-scale Cloud computing system is divided into multiple subsystems to form a multi-level Cloud computing system (hierarchical Cloud computing with multi subsystems (HCCMS)). Similar to but different from the Refs. Lu et al. (2020) and Li et al. (2020), Fig. 2 presents one architecture of HCCMS.

In Fig. 2, a user or a group of users submits diverse requests to Cloud system, the classifier in Cloud center will classify these requests and send the classified requests to the corresponding subsystems for processing. Then the management center (scheduler) of the subsystem allocates these requests to specific physical or virtual machines. This HCCMS structure in Fig. 2 reduces the overall failure probability of system and simultaneously increases the security between subsystems. In case of local subsystem failure, the management center of other subsystems or Cloud center can temporarily take over the resource scheduling of the failed subsystem under the authorization of the Cloud system management center. In order to meet more complex scenarios during operation of Cloud computing, the nodes in all subsystems can be re-divided and combined into new subsystems dynamically. HCCMS also indicates that the resource scheduling strategies of different subsystems may be different, which poses a great challenge to Cloud.

Focusing on resource scheduling problems in HCCMS, we consider a HCCMS with q subsystems donated as $S = \langle S_1, S_2, \ldots, S_q \rangle$ and the kth subsystem with n_k server nodes donated as $N_k = \langle N_{k1}, N_{k2}, \ldots, N_{kn_k} \rangle$. As each server node has a limited capacity of CPU and disk storage (DS), we use C_{ki}^{CPU} and C_{ki}^{DS} to denote them. It can be assumed that the set of tasks allocated to the subsystem S_k at a time slot $[t, t+\Delta t)$ as $T_k = \langle T_{k1}, T_{k2}, \ldots, T_{km_k} \rangle$ with m_k tasks. Then, we use v_{kji}^{CPU}, and v_{kji}^{DS} as the capacity request of task T_{kj}, ET_{kji} as the processing time of T_{kj}, as well as L_{ki}^{CPU} and L_{ki}^{DS} as the load of R_{ki}.

In order to illustrate the advantages of HCCMS, we theoretically analyze the computational complexity of the resource scheduling algorithm in HCCMS. Supposing the computational complexity of an algorithm to allocate m tasks to n resources is $O(f(m,n))$. For the existing resource scheduling algorithms, except that algorithms with linear complexity such as random algorithm, other algorithms have polynomial or exponential computational complexities and meet the $f''_m(m,n) \geq 0$ and $f'_n(m,n) \geq 0$. Therefore, Eq. (1) can be obtained based on Jensen Inequality.

$$f(m,n) = f(m,n) + f(0,n) \geq \sum_{k=1}^{q} f(m_k, n) \geq \sum_{k=1}^{q} f(m_k, n_k) \tag{1}$$

Especially, when an algorithm has polynomial complexity above quadratic or exponential complexity, the computational complexity satisfies $\exists \alpha \geq 2$ **s.t.** $O(f(m,n)) \geq O(m^{\alpha})$. Then, Eq. (2) can be obtained based on Power-Mean Inequality.

$$f(m,n) \geq q^{(\alpha-1)} \sum_{k=1}^{q} f\left(\frac{m}{q}, n\right) \geq q^{(\alpha-1)} \sum_{k=1}^{q} f\left(\frac{m}{q}, n_k\right) \tag{2}$$

According to Eq. (2), evenly allocating the tasks to subsystems can reduce the computational complexity of resource scheduling to $q^{1-\alpha}$ times. Eqs. (1) and (2) demonstrate the significance of HCCMS in improving resource management capability.

With the theoretical analysis of dividing Cloud system into multi subsystems, we can continue to construct its system model. Assuming a server node (PM or VM) is only in one subsystem, we consider a subsystem has two basic classifications i.e. homogeneous and heterogeneous. The homogeneous subsystem means the execution of user's task requests on each node is the same as well as the upper load limit of each node is the same, whose relationship can be shown as Eq. (3). Otherwise, the subsystem is heterogeneous.

$$\begin{cases} \left\langle v_{kji_1}^{CPU}, v_{kji_1}^{DS} \right\rangle = \left\langle v_{kji_2}^{CPU}, v_{kji_2}^{DS} \right\rangle, \\ ET_{kji_1} = ET_{kji_2}, \\ \left\langle C_{ki_1}^{CPU}, C_{ki_1}^{DS} \right\rangle = \left\langle C_{ki_2}^{CPU}, C_{ki_2}^{DS} \right\rangle, \\ 1 \leq i_1 < i_2 \leq n_k, 1 \leq \forall j \leq m_k. \end{cases} \tag{3}$$

In Cloud computing, the server nodes, commonly used to process the same type of tasks, may approximately be homogeneous. For example, the disks dedicated to the service nodes that processes storage requests may be the same batch of production with the same model and for storage requests, if disks are identical then the server nodes can be regarded as homogeneous. Additionally, the server nodes, which are frequently applied to process some computing requests or comprehensive requests, may be heterogeneous because computing requests and comprehensive requests from different users vary greatly.

In this paper, we consider the task requests are integral and cannot be split into smaller ones, which means any task will be fully allocated to only one server node, however one server node may process more than one task simultaneously. We denote the set of tasks in node R_{ki} as TS_{ki} where if a task T_{kj} is allocated to R_{ki}, then $T_{kj} \in TS_{ki}$. All the TS_{ki} in subsystem S_k constitute a vector $KS_k = \langle TS_{k1}, TS_{k2}, \ldots, TS_{kn_k} \rangle$. As KS_k determines the unique allocation result of tasks in subsystem S_k, it can be leveraged to represent the solution of tasks allocation of subsystem S_k. According to the occupation status of the task or VMs on the server nodes, it can be divided into two categories: a task occupies the server node completely within a certain time (full-occupancy),

and the task occupies part of the server node (partial-occupancy). If a task occupies the server node completely, the server node can only process one task simultaneously (Ghalami and Grosu, 2019; Ding et al., 2020), which implies other tasks need to enter the queue and wait for the server node to be idle before being processed. For this situation, running time of a server node RT_{ki} is the main parameter to be considered, and the total running time of the server node is equal to the sum of the processing time of all tasks assigned to this node as Eq. (4).

$$RT_{ki} = \sum_{T_{kj} \in TS_{ki}} ET_{kji} \tag{4}$$

where we consider the processing time of tasks is fixed without affection from the resources' status or the order of tasks. For partial-occupancy of server node, running time is no longer the superposition of tasks' processing time. However, the occupation of various components in the server node is a more noteworthy parameter. Then, we consider the tasks currently allocated to the server node will occupy the resources of the node simultaneously in a minimum time slot. As the occupancy of components in the server node generally satisfies the relationship of linearly superposition, Eq. (5) can be given

$$L_{ki}^{CPU} = \sum_{T_{kj} \in TS_{kj}} v_{kji}^{CPU} \tag{5}$$

where the similar relationship applies to DS.

In this paper, we do not address the issue of multi-dimensional resource scheduling, so assume each scheduler only needs to consider the one-dimensional resource. Finally, multiple factors, that are the homogeneity and heterogeneity of server nodes, full-occupancy and partial-occupancy will construct different types of subsystems in Cloud computing, hence increasing the abilities of Cloud systems to provide flexible services.

3.2. Subsystems and subproblems of resource scheduling

Based on the types of tasks requests and services, several types of the subsystem in HCCMS can be presented as:

(1) Homogeneous full-occupied subsystem, which is usually utilized to provide specific services for user groups with exclusive needs considering that these users require independence and stability during the service period and the type of their tasks are stationary;
(2) Homogeneous partial-occupied subsystem, which is usually utilized to provide services for the broader user groups with a small capacity of the single task but the huge number of all task requests, such as dedicated subsystem for file storage supporting Cloud disk;
(3) Heterogeneous full-occupied subsystem, which is usually utilized to provide comprehensive or customized services for specific user groups that these users require to independently occupy the allocated server nodes and the types of their tasks are more complex, such as some server nodes for commercial or scientific research activities;
(4) Heterogeneous partial-occupied subsystem, which is constructed with various physical machines and usually utilized to provide comprehensive services for broader users.

For the full-occupied subsystem, the running time applied by users is the main parameter affecting the service capability of server nodes. Thus, optimizing the total running time of the server node is one of the keys to manage the full-occupied subsystem.

For the homogeneous full-occupied subsystem, the processing time of a task is the same for different server nodes, so the total running time of all the server nodes in the homogeneous full-occupied subsystem is invariant when tasks are given. Then, an optimization problem to minimizing the makespan of the homogeneous full-occupied subsystem can be given as Eq. (6), where the makespan means the maximum running time of all server nodes in subsystem S_k, and the constraints are shown as Eq. (7).

$$\min \omega_k^{(1)} = \min \left(\max_{i=1,2,\ldots,n_k} \left(\sum_{j=1}^{m_k} x_{ji} ET_{kji} \right) \right) \tag{6}$$

$$\mathbf{s.t.} \begin{cases} \sum_{i=1}^{m} x_{ji} = 1, \sum_{j=1}^{n} x_{ji} v_{kji} \le C_{ki}, \\ x_{ji} \in \{0,1\}, j \in \{1,2,\ldots,m_k\}, i \in \{1,2,\ldots,n_k\} \end{cases} \tag{7}$$

For heterogeneous full-occupied subsystem, the processing time of the same task may vary on heterogeneous server nodes, which indicates different schemes KS_k of tasks allocation will lead different total running time of subsystem. Therefore in this case, the total running time and makespan are both critical optimization objectives shown as Eq. (8) also subject to Eq. (7).

$$\min \omega_k^{(2)} = \begin{cases} \min \left(\max_{i=1,2,\ldots,n_k} \left(\sum_{j=1}^{m_k} x_{ji} ET_{kji} \right) \right) \\ \min \left(\sum_{i=1}^{n_k} \sum_{j=1}^{m_k} x_{ji} ET_{kji} \right) \end{cases} \tag{8}$$

For homogeneous partial-occupied subsystem, we consider two types of subsystems classified by their services: one is the subsystem applied to storage requests and the other is that applied to computing requests.

The main component for Cloud storage requests is hard disk which mainly requires load balancing to reduce network congestion. In this paper, we use variance of server nodes' load of disk storage to measure the degree of balancing. Next, the objective of load balancing can be given as

$$\begin{aligned} \min \omega_k^{(3)} = \min & \frac{1}{n_k} \sum_{i=1}^{n_k} \left(\sum_{j=1}^{m_k} x_{ji} v_{kji}^{DS} \right)^2 \\ & - \frac{1}{n_k^2} \left(\sum_{i=1}^{n_k} \left(\sum_{j=1}^{m_k} x_{ji} v_{kji}^{DS} \right) \right)^2 \end{aligned} \tag{9}$$

where the constraints are as Eq. (7) and $\sum_{j=1}^{m_k} x_{ji} v_{kji}^{DS} \le C_{ki}^{DS}$.

The computing requests mainly rely on CPU or GPU which usually consume more electrical energy than disk, hence one of the objectives is to reduce the number of working nodes of subsystem (bin packing problem). In this paper taking the utilization of the CPU as an instance, the single-dimensional bin packing problem can be given as

$$\min \omega_k^{(4)} = \min \sum_{i=1}^{n_k} \max_{j=1,2,\ldots,m_k} x_{ji} \tag{10}$$

where the constraints are as Eq. (7) and $\sum_{j=1}^{m_k} x_{ji} v_{kji}^{CPU} \le C_{ki}^{CPU}$.

For the heterogeneous partial-occupied subsystem providing comprehensive services for broader user groups, the number of instructions accounts for a high proportion of its tasks to support various tasks. Thus, the CPU is one of the most frequently used components. In this case, balancing the utilization of the CPU and minimizing the total occupied capacity of the CPU are two considerable objectives, hence a bi-objective optimization problem can be given as

$$\min \omega_k^{(5)} = \begin{cases} \min \frac{1}{n_k} \sum_{i=1}^{n_k} \left(\sum_{j=1}^{m_k} x_{ji} v_{kji}^{CPU} \right)^2 \\ \quad - \frac{1}{n_k^2} \left(\sum_{i=1}^{n_k} \left(\sum_{j=1}^{m_k} x_{ji} v_{kji}^{CPU} \right) \right)^2 \\ \min \sum_{i=1}^{n_k} \sum_{j=1}^{m_k} x_{ji} v_{kji}^{CPU} \end{cases} \tag{11}$$

According to the properties of different subsystems, we have given five subproblems of HCCMS summarized in Table 2. Although there

Table 2
Five subproblems in this paper.

Sign	Description of problem
$\omega^{(1)}$	Minimizing makespan for homogeneous resources
$\omega^{(2)}$	Minimizing makespan for heterogeneous resources
$\omega^{(3)}$	Load balancing of DS for homogeneous resources
$\omega^{(4)}$	Bin packing for homogeneous resources
$\omega^{(5)}$	Minimizing standard deviation and total load of CPU for heterogeneous resources

are still more optimization problems for Cloud computing not listed in Table 2, the core target of this paper is to address the joint scheduling problems of HCCMS with various subproblems. Thus, using these five optimization problems as examples has representativeness actually. For the joint scheduling of more types of scheduling problems, the proposed method in this paper will still be applicable.

3.3. Joint scheduling problem and cost model for various subproblems

In this paper, we define the joint scheduling problem for various subproblems as optimization problems with variable optimization objectives where the optimization objectives are known before the scheduling of resources. Similar to Gudu et al. (2018), we also construct cost model $U_k(t)$ for the trade-off the solution quality $\omega_k(t)$ and computational complexity $\tau_k(t)$ of scheduling algorithm at the time slot $[t, t+\Delta t)$ to uniformly measure the solutions of various subproblems. Without losing generality, we assume the cost equals the weighted sum of optimization results and computational complexity using two weights $w_\omega^{(k)}(t)$ and $w_\tau^{(k)}(t)$ as

$$U_k(t) = w_\omega^{(k)}(t) \cdot \omega_k + w_\tau^{(k)}(t) \cdot \tau_k \tag{12}$$

where $w_\omega^{(k)}(t), w_\tau^{(k)}(t) \in \mathbb{R}$. Considering the complexity of realistic scenarios, we regard the weights $w_\omega^{(k)}(t)$ and $w_\tau^{(k)}(t)$ as time-varying for each subsystems. Generally, the objective of minimizing the cost $\Omega(t)$ of the whole system at the time slot $[t, t+\Delta t)$ can be given as

$$\min \Omega(t) = \min \sum_{k=1}^{q} U_k(t) \tag{13}$$

4. Methodology: Selectors of scheduling algorithms in HCCMS

4.1. SFSSA: Scheduling framework to select the scheduling algorithms

For the joint scheduling problem for various subproblems as Eq. (13), a single algorithm is not enough to deal with varying and complex scenarios, which proposes a demand for flexible utilization of various algorithms regarding scheduling algorithms as selectable tools. Naturally, the joint scheduling problem for various subproblems can be converted to: *how to select an appropriate algorithm to schedule resources of a subsystem aiming to minimize total cost of HCCMS*.

Obviously, the most appropriate algorithm corresponding to different weights may be non-fixed. For example, when $w_\omega^{(k)}(t) > 0 \wedge w_\tau^{(k)}(t) = 0$ or $w_\omega^{(k)}(t) \gg w_\tau^{(k)}(t) \geq 0$, an algorithm with high complexity but can get the optimal solution is the most appropriate; contrarily, when $w_\omega^{(k)}(t) = 0 \wedge w_\tau^{(k)}(t) > 0$ or $w_\tau^{(k)}(t) \gg w_\omega^{(k)}(t) \geq 0$, a fast algorithm such as greedy algorithm or random algorithm may be the most appropriate. It can be assumed an algorithm pool $A = \langle A_1, A_2, \ldots, A_d \rangle$ has d algorithms and the optimization result and computational complexity of lth algorithm for the subproblem at the time slot $[t, t+\Delta t)$ of the subsystem S_k are respectively $\omega_{kl}(t)$ and $\tau_{kl}(t)$. If an algorithm A_l cannot solve the subproblem of the subsystem S_k, we set $\omega_{kl}(t) = +\infty$ and $\tau_{kl}(t) = +\infty$. Then, Eq. (13) can be transformed into Eq. (14) based on: *selecting an appropriate algorithm*, where the constraints are as Eq. (15).

$$\min \Omega(t) = \min \sum_{k=1}^{q} \sum_{l=1}^{d} y_{kl}(t) \left(w_\omega^{(k)}(t)\omega_{kl}(t) + w_\tau^{(k)}(t)\tau_{kl}(t) \right) \tag{14}$$

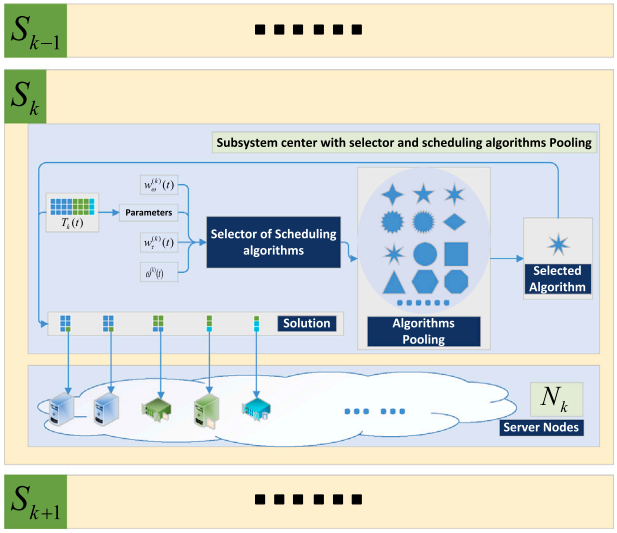


Fig. 3. The scheduling framework to select the scheduling algorithms (SFSSA).

$$\text{s.t.} \begin{cases} \sum_{l=1}^{d} y_{kl}(t) = 1, y_{kl}(t) \in \{0,1\}, \\ k \in \{1,2,\ldots,q\}, l \in \{1,2,\ldots,d\} \end{cases} \tag{15}$$

On the premise of given weights and tasks of all subsystems, the center of HCCMS needs to select a suitable algorithm for each subsystem to minimize the total cost of the system. As the scheduling algorithms can be deployed by multiple subsystems simultaneously, we only need to construct one public algorithms pool. Then, the scheduling framework to select the scheduling algorithms (SFSSA) from the algorithms pool to schedule the resources of the subsystem is as Fig. 3, and its algorithm is as Algorithm 1.

Algorithm 1: Scheduling framework to select the scheduling algorithms (SFSSA)

Input : Tasks set $T_k(t)$ and their parameters, objective of scheduling problems $\omega^{(k)}(t)$, weights of cost $w_\omega^{(k)}$ and $w_\tau^{(k)}$, and algorithms pool A
Output: Selected algorithm A_y also the solution $\langle y_{k1}(t), y_{k2}(t), \ldots, y_{kq}(t) \rangle$ of problem Eq (14)
1 Use algorithm selector to select an algorithm from algorithms pool
2 Execute the selected algorithm to generate the solution of scheduling problem
3 Schedule the resources based on the solution
4 Calculate the cost and update the strategies of selector

In Eq. (14), if solution $\omega_{kl}(t)$ and complexity $\tau_{kl}(t)$ are given before scheduling, it only needs to select the algorithm with the minimum cost $w_\omega^{(k)}(t)\omega_{kl}(t) + w_\tau^{(k)}(t)\tau_{kl}(t)$. However, before executing a scheduling algorithm, $\omega_{kl}(t)$ and $\tau_{kl}(t)$ are usually unknown. Thus, the directly required function of algorithm selector is to predict $\omega_{kl}(t)$ and $\tau_{kl}(t)$, which nevertheless is difficult. Considering this issue, we convert the prediction of $\omega_{kl}(t)$ and $\tau_{kl}(t)$ to the classification according to the input of Algorithm 1 and utilize a classifier to act as the algorithm selector of SFSSA. Now, we can see that two main elements of SFSSA are actually classifier (i.e. algorithm selector) and algorithms pool.

4.2. Algorithms pool

Focusing on the five subproblems presented in Table 2, we construct an algorithms pool with various algorithms including heuristic

Table 3
Various types of algorithms in algorithms pool.

Category	Algorithm	Description	Subproblems				
			$\omega^{(1)}$	$\omega^{(2)}$	$\omega^{(3)}$	$\omega^{(4)}$	$\omega^{(5)}$
Randomization	Random	Randomly allocating tasks to resources	✓	✓	✓	✓	✓
Heuristic	Greedy	Scheduling with greed priory	✓	✓	✓	✓	✓
	RR	Round Robin algorithm	✓	✓	✓	×	✓
	LPT	Longest processing time algorithm	✓	×	✓	×	×
	SPT	Shortest processing time algorithm	✓	✓	✓	×	✓
	BFD	Best fit decreasing algorithm	✓	×	✓	✓	×
	FFD	First fit decreasing algorithm	×	×	×	✓	×
	FF	First fit algorithm	×	×	×	✓	×
Meta-heuristic	ACO	Ant colony optimization algorithm	✓	✓	✓	✓	✓
	PSO	Particle Swarm optimization algorithm	✓	✓	✓	✓	✓
	GA-Random	Genetic algorithm with random initial state	✓	✓	✓	✓	✓
Single routelocal search	LPTS	LPT-Search algorithm using LPT as search route	✓	✓	✓	×	✓
	MLPTS	MLPT-Search algorithm using MLPT as search route	✓	✓	✓	×	✓
	BFDS	BFD-Search algorithm using BFD as search route	✓	×	✓	✓	×
	NS	Neighborhood-Search algorithm	✓	×	✓	✓	×
	BestBFDS	BestBFD-Search using BestBFD as search route	✓	✓	✓	✓	✓
Multi routeslocal search	LPT-NS	Using LPT and NS as search routes	✓	✓	✓	×	✓
	MLPT-NS	Using modified LPT and NS as search routes	✓	✓	✓	×	✓
	BFD-NS	Using BFD and NS as search routes	✓	×	✓	✓	×
	LPT-BFD-NS	Using the LPT, BFD and NS as search routes	✓	×	✓	×	×
Hybrid	GA-MinMin	Genetic algorithm using MinMin initialized state	✓	✓	✓	✓	✓
	ACO-GA	Using the output of ACO as the input of GA	✓	✓	✓	✓	✓
	PSO-GA	Using the output of PSO as the input of GA	✓	✓	✓	✓	✓
	MLPT-GA	Using GA and MLPT as search routes	✓	✓	✓	✓	✓
Machine learning	DRL	Deep reinforcement learning	✓	✓	✓	✓	✓

algorithms, meta-heuristic algorithms, local search algorithms, hybrid algorithms and deep reinforcement learning. The specific algorithm and its corresponding subproblems are shown in Table 3. In Table 3, LPTS, MLPTS, BFDS, BestBFDS, LPT-NS, MLPT-NS, BFD-NS and LPT-BFD-NS are all heuristic-based local search algorithms utilizing heuristic algorithms as search routes. The algorithm of the heuristic-based local search algorithm is shown as Algorithm 2, which can significantly reduce the computational complexity and enhance the optimality of algorithms. Among Table 3, MLPTS and BestBFDS are modified algorithms originating from LPT and BFD respectively to solve subproblems, as well as their search routes are shown as Algorithm 3 and Algorithm 4.

Algorithm 2: Heuristic-based local search algorithm

Input : Subproblem ω_k and the set of tasks T_k
Output: Solution KS_k of the subproblem
Initially Allocate tasks to resources with confirmed or random initialization policy and gain the general initial status of KS_k
while Exists_Ner **do**
  NoExists_Ner
  Search neighborhood $Ner(KS_k)$ of KS_k in the heuristic-based local search algorithm
  if $KS'_k \in Ner(KS_k)$ ***s.t.*** *the solution of* KS'_k *is optimal than* KS_k **then**
    Exists_Ner
    Choose the optimal neighbor to update $KS'_k \rightarrow KS_k$
Set KS_k as the solution of ω_k

4.3. DLS: DL-based selector of scheduling algorithms

After giving algorithms pool, one of method to select appropriate algorithms is Deep Learning-based selector (DLS). As deep learning belongs to supervised learning, a labeled dataset is required to train the DLS. To build a labeled dataset, we randomly generate these five categories of problems through the simulation system; execute all algorithms in the algorithms pool suitable for the problem; then record the solutions and complexities of these scheduling algorithms; lastly sort the cost of each algorithm from small to large, and set the label value according to the sorting ranking.

Algorithm 3: Modified LPT search route for heterogeneous resources

Input : Tasks $TS = TS_{ki} \cup TS_{kl}$ of R_{ki} and R_{kl}
Output: TS_{ki} and TS_{kl}
$Mark_i = 0$, $Mark_l = 0$, $TS_{ki} = \emptyset$ and $TS_{kl} = \emptyset$
while $TS \neq \emptyset$ **do**
  if $Mark_i \leq Mark_l$ **then**
    $c = i$, $b = l$
  else
    $c = l$, $b = i$
  Find tasks $T_{k\alpha_o} \in TS$ **s.t.** $E_{k\alpha_o} = \min_{T_{kj} \in TS} \left(ET_{kjc} - ET_{kjb}\right)$ to obtain a set of $\{T_{k\alpha_1}, T_{k\alpha_2}, \ldots, T_{k\alpha_s}\}$
  if $s \geq 2$ **then**
    Choose $T_{k\alpha}$ **s.t.** $ET_{k\alpha} = \max_{1 \leq o \leq s} ET_{k\alpha_o c}$
  $Mark_c += E_{k\alpha c}$, $TS_{kc} += \{T_{k\alpha}\}$ and $TS -= \{T_{k\alpha}\}$

For DLS, we establish several deep models with full connections as Table 4. Tasking 100 000 training data and 10 000 testing data, the accuracy and loss in training progress of these DLSs are plotted in Fig. 4. From Fig. 4, the test accuracies of these three DLSs are about 95% after training. Without losing generality, we choose the structure of DN_3 as the network structure of subsequent DRLSs and use well-trained DN_3 as their pre-trained model.

4.4. DRLS: DRL-based selector of scheduling algorithms

DLS depends on the establishment of the labeled dataset marking the current best algorithm. During the operation of the actual Cloud computing system, it is unknown in advance which resource scheduling

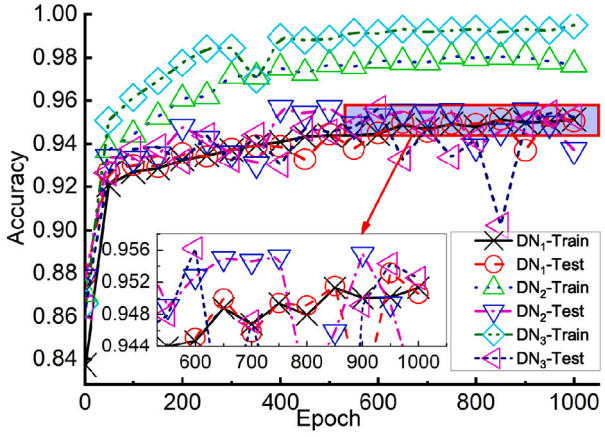
Algorithm 4: BestBFD search route

Input : Tasks $TS = TS_{ki} \cup TS_{kl}$ of R_{ki} and R_{kl}
Output: TS_{ki} and TS_{kl}

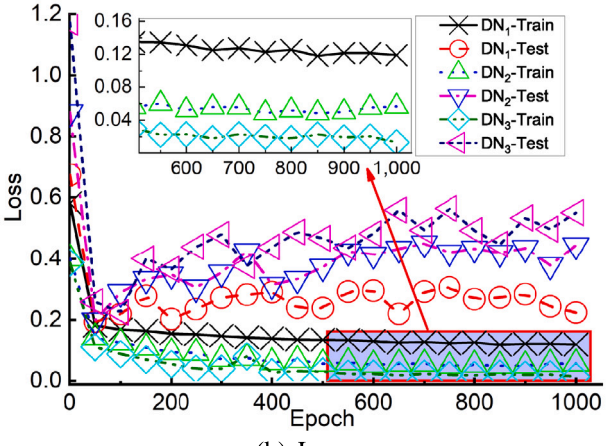
Set $\gamma = \frac{\sum_{T_{kj} \in TS_{ki} \cup TS_{kl}} ET_{kji}}{2}$, $s = 0$ and $Mark_i = 0$
for *j in tasks from largest to smallest* **do**
  if $Mark_i + ET_{kji} \leq \gamma$ **then**
    Put task in R_{ki} and update $Mark_i += ET_{kji}$
  else
    $s++$
    Mark $\left|Mark_i + E_{kji} - \gamma\right| = L_s$
    Mark the scheme as KS_s
Mark the scheme as KS_0 and Choose $\arg\min_{KS} \left(\left|Mark_i - \gamma\right|, L_1, \ldots, L_s\right)$ as solution

Table 4
DNN structures of DLSs.

Sign	Number of layers	Structure (with full connections)
DN_1	3	64 → 128 → 25
DN_2	4	64 → 128 → 256 → 25
DN_3	5	64 → 128 → 256 → 512 → 25



(a) Accuracy.



(b) Loss.

Fig. 4. The train process for DL-based selectors of Table 4.

algorithm is most suitable for the dynamic scenarios. Therefore, the other one method able to act as algorithms selector is Deep Reinforcement Learning-based selector (DRLS). Based on the decision between exploitation and exploration, the advantages of reinforcement learning are to support non-supervised learning or semi-supervised learning, as well as to support lifelong learning. The utilization of DNN in RL is conducive to adapting to the continuous input space of SFSSA. In this paper, the base components of DRLS are defined as:

(1) Environment: The environment consisting of a simulated system receives the actions from a agent (or agents) of DRLS, executes the selected scheduling algorithms, then calculates the cost according to the performance of selected algorithms and feedback the reward of selected algorithms.
(2) State space: The state space refers to the current status of each subsystem including the objective of subproblems, weights of cost and tasks set, which are the input of DRLS.
(3) Action space: The action space corresponding to the algorithms pool means selecting a specific algorithm to address the corresponding subproblem.
(4) Policy: The policy is a decision maker and also the selection policy for resource scheduling algorithms in this paper.
(5) Reward and Loss Function: Training DRLS requires some feedback from the environment. In this paper, we use two types of feedback to train DRLS with random alternation to accelerate the training process: one is the reward function equal to the reciprocal of the cost i.e. $1/U_k(t)$ and train DRLS with policy gradient shown in Algorithm 5; the other is using the best selection among some given strategies and a target net as the label to get the cross-entropy loss and train DRLS by gradient descent. The principle of the combination of two types of feedback is two different gradient routes can jump out of the local convergence of a single route.

Algorithm 5: Update parameters by policy gradient

Initialize parameters of networks θ arbitrary or inherit them from DL-based selector
for *the experience of each episode* $\langle s(1), a(1), r(1)\rangle, \ldots, \langle s(t), a(t), r(t)\rangle$ π_θ **do**
  for $i = 1 \rightarrow t$ **do**
    $\theta \rightarrow \theta + \alpha \nabla_\theta \log \pi_\theta \left(s_i, a_i\right) v_i$
return θ

For HCCMS, the policy gradient is as:

$$g = \sum_{k=1}^{q} \left\{ q_\pi \left(s_{tk}, a_{tk}\right) \nabla_\theta \sum_{i=0}^{t} \log \pi \left(a_{ik} | s_{ik}; \theta\right) \right\} \tag{16}$$

and the updated parameters are as:

$$\theta = \theta + \alpha g \tag{17}$$

where $q_\pi \left(s_{tk}, a_{tk}\right)$ is the score for subsystem S_k based on policy π_θ, $\pi \left(a_{ik} | s_{ik}; \theta\right)$ is the conditional probability of action a_{ik} under status of s_{ik} of subsystem S_k based on current policy π_θ, and $\sum_{k=1}^{q} \sum_{i=0}^{t}$ means using accumulate feedback of all subsystems to update parameters of DRLS.

The other method to get the gradient of training DRLS is target labels of action. Combining the preponderance of DDQN and DL, we choose the best action of the following strategies as the target action.

(1) DLS: a well-trained DLS based on the labeled dataset;
(2) Delayed network of DRLS: the DRLS saved regularly during the training process;
(3) Random selector: randomly selecting one algorithm;
(4) Fixed selector: fixedly selecting one or several algorithms;

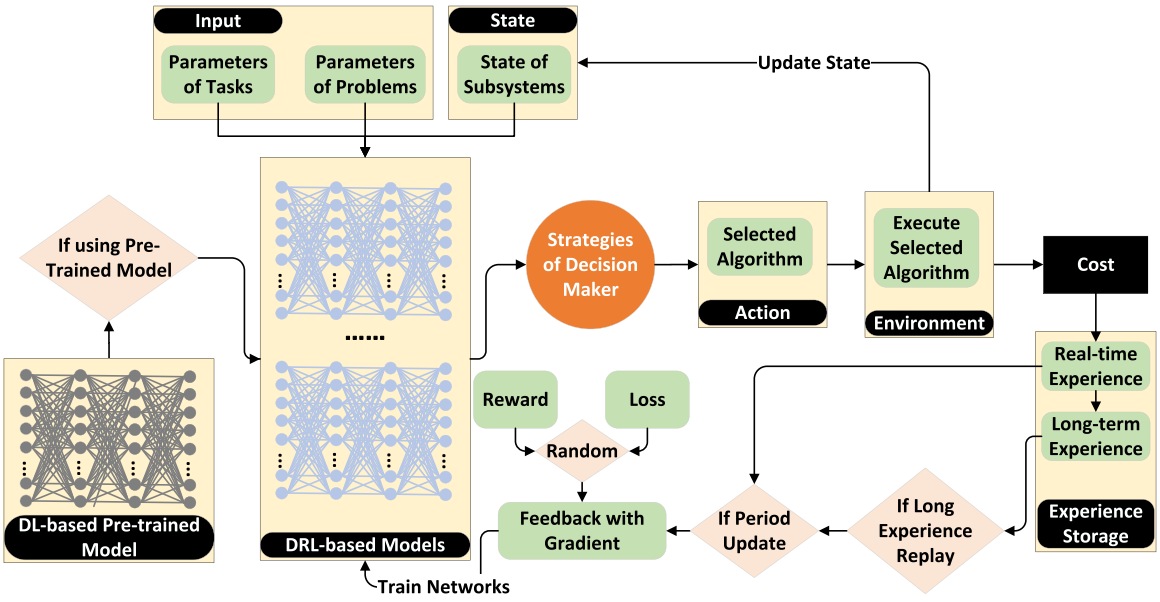


Fig. 5. A framework of DRL-based selector with various strategies.

Algorithm 6: Training of DRLS with multi strategies

Initialize parameters of DRL-based networks (randomly initialize or inherit parameters from DL-based selector)
for $time = 0 \rightarrow t$ **do**
  Generate the objective of subproblem and the weights $w_{\omega}^{(k)}(time)$ and $w_{\tau}^{(k)}(time)$
  Generate parameters of tasks
  Input and obtain selection results of DL-based selector, delayed network of trained selector, random selector, fixed selector and DRL-based selector at *time* respectively
  Execute selected algorithms of above selectors and calculate costs based on $w_{\omega}^{(k)}(time)$ and $w_{\tau}^{(k)}(time)$
  Get the best selection result or ranking according to costs
  Get the reward of the selection result according to the ranking or get the label according to the best selection result
  Store reward or label into experience pool
  if *replay long-term experience* **then**
    Randomly selected long-term experience from experience pool
  if *period update* **then**
    Train the networks of DRL-based selector according to real-time experience and long-term experience
  if *use delayed network and reach the periodic save point* **then**
    Save current network as delayed network

(5) DRLS itself: One or several DRLS with their own results.

In order to verify and select the appropriate DRLS model, we train multiple selectors based on different strategies. The strategies of DRLS considered in this paper contain:

(1) Pre-trained model: using the model trained by DLS with labeled dataset as the initial network of DRLS;
(2) Long term experience replay & periodical update (LTPU): the environment stores the experience $e(t) = \langle s(t), a(t), r(t) \rangle$ into replay memory $D(t) = \langle e(1), e(2), \ldots, e(t) \rangle$, then randomly selects the experiences in the experience memory at a specific time to participate in the training of DRLS;
(3) Joint training: multi-agents of DRL participate in training and generation of target value simultaneously.

With these components, a framework of DRL-based selector can be shown as Fig. 5. The framework of Fig. 5 contains multiple components of strategies and the algorithm of training DRLS is shown in Algorithm 6. However, a DRL-based model may only utilize some of them and different combinations of them have distinctness in performance. Leveraging various training strategies and their combinations, we train and obtain five DRL-based models as Table 5.

Table 5
DRL-based models combining various strategies trained in this paper.

Models	Pre-trained	LTPU	Joint training
$Model_1$	×	×	×
$Model_2$	×	✓	×
$Model_3$	✓	×	×
$Model_4$	×	✓	✓
$Model_5$	✓	✓	✓

5. Performance evaluation

5.1. Experiment setting

Currently, there is limited literature exploring the selector of the resource scheduling algorithm in Cloud computing and some algorithm

Table 6
Compared baseline strategies.

Strategies	Description
RS (Gudu et al., 2018)	Random selector: randomly select an algorithm
GS (Gudu et al., 2018)	Greedy selector: greedily select the algorithm with solution quality or complexity
RRS	Round-robin selector: select each algorithm in turn
SBS (Seiler et al., 2020; Deshpande et al., 2021)	Single best selector: select the single algorithm with best average performance
VBS (Seiler et al., 2020; Deshpande et al., 2021)	Virtual Best Selector: perfectly select the algorithm with the best previous statistical cost
SFS	Single fast selector: select the fastest algorithm

selection strategies in other research directions contain Random Selector (RS) (Gudu et al., 2018), Greedy Selector (GS) (Gudu et al., 2018), Single Best Solver (SBS) (Seiler et al., 2020; Deshpande et al., 2021), Virtual Best Solver (VBS) (Seiler et al., 2020; Nguyen et al., 2020), Optimal Decision Tree (Boas et al., 2021) and Machine Learning (Muñoz and Kirley, 2021). Among these, RS, GS, VBS and SBS are most commonly used as baselines of algorithms selection. Adding two more selection strategies including Round-Robin Selector and SFS (select the fastest single algorithm), we construct the compared baselines of this paper as Table 6.

With baselines in Table 6, DLSs in 4 and DRLSs in 5 to evaluate the performances of DLS and DRLS, we carry out four sets of comparative experiments respectively between:

(1) EX_0: Cloud system without subsystem and HCCMS;
(2) EX_1: Well-trained DLSs in Table 4 and baseline strategies in Table 6;
(3) EX_2: Dynamically trained DRLS ($Model_1$) and baseline strategies;
(4) EX_3: Dynamically trained $Model_1$ to $Model_5$ in Table 5 (various strategies of DRLSs).

These sets of experiments are all based on the variable-controlling approach. Among them, EX_0 fixes specific scheduling problems and changes the number of subsystems to observe the influence of the number of subsystems on the computational complexity and optimization results of several scheduling algorithms. Fixing HCCMS structure, parameters of tasks and server nodes, and parameters of cost model, EX_1, EX_2 and EX_3 are to change algorithm selection strategies so as to observe the impact of different algorithm selection strategies in various scenarios. EX_1 is to evaluate the performance of DLSs compared to baselines in the scenarios with relative static ranges of parameters and simultaneously to test the adaptability of DLSs for the scenarios outside the training dataset. EX_2 is to evaluate the performance of DRLS compared to baselines in the relative dynamic scenarios. And then, EX_3 is to evaluate the performance of different strategy combinations in DRLS. The combination of these four sets of experiments illustrates the advantages of HCCMS structure in resource scheduling, illustrates the advantages of DLS and DRLs in specific scenarios, and finally provides a reference for the choice of algorithm selectors.

Subsystem type often determines its user type, optimization objective and characteristic parameter. Additionally, different characteristic parameters have different degrees of effects on the cost model. Therefore, we set different parameter ranges according to the characteristic parameters of subsystems for various objectives, which can be seen in Table 7.

Then, we simulate the HCCMS through a Python-based simulation environment and launch the experiments on a desktop computer with configurations as follows:

- CPU: Intel(R) Core(TM) i5-8400 CPU @ 2.8 GHZ.
- SSD: KINGSTON SA400S37 240 GB.
- GPU: NVIDIA GeForce GTX 1060 6 GB.
- Program version: Python 3.6, Tensorflow.

5.2. Results and discussion

5.2.1. EX_0: Single-layer system vs. Multi-layer system

Before experiments of verifying algorithms selectors, we carry out a set of experiments to compare the traditional single-layer system and multi-layer system (HCCMS). To facilitate comparison in this set of experiments, we set the scheduling problem as minimizing makespan for heterogeneous resources (i.e. $\min \omega^{(2)}$). We use three algorithms that Random algorithm, MLPTS and GA (with fixed 600 generations and 100 populations) for verification of complexities and optimization results. We experiment in five types of system organizations for the same tasks set with 10^4 tasks waiting to be allocated. The parameters of tasks are consistent with those of $\min \omega^{(2)}$ in Table 7 and not subject to the constraints, which means each task is randomly generated by the uniform distribution $ET_{kji} \sim \mathrm{U}(30, 120)$ min. Using the number of subsystems as the main variable, these five types of system organizations are respectively no subsystem (corresponding to abscissa 1), 5 subsystems, 10 subsystems, 25 subsystems and 50 subsystems. In the system organizations with subsystems, we divide the tasks into the number of subsystems and each subsystem processes the same number of tasks. Then, the results of two experiments respectively with 100 server nodes and 200 server nodes are plotted in Fig. 6.

Fig. 6(a) plots the complexities and makespan for the system with 100 server nodes and Fig. 6(b) for that with 200 server nodes. From Fig. 6, the complexities of MLPTS and GA both decrease with the increase of the number of subsystems. This validates the relationship of Eq. (2) that dividing the Cloud system into multi-layer systems with multiple subsystems can significantly reduce the complexities of resource scheduling. The computational complexity of the random algorithm does not vary with the change of system organizations, which is because the random algorithm does not consider the characteristics of the task itself, and its computational complexity is equal to the total number of tasks. The makespan of MLPTS increases with the increase of the number of subsystems, while that of GA decreases slightly and that of random algorithm overall increases but with instability. The increase of makespan using MLPTS algorithm is caused by HCCMS structure where the solution of MLPTS algorithm is close to the global optimal solution but the uneven assignment of tasks to each subsystem leads to the increase of global optimal makespan. This reason can also explain the overall increasing trend of the makespan of random algorithm. Additionally about the decreasing trend of GA, the solution of GA is not close to the global optimal solution and the solution search space of multiple subsystems is far smaller than that of no subsystem, so GA with fixed generations and populations can find a relatively better solution in the scenario of multiple subsystems than that of no subsystem. Specifically discussing 100 server nodes in Fig. 6(a), the complexity of MLPTS in no subsystem is 575M and that in 5 subsystems is 35.9M, as well as the makespan of MLPTS in no subsystem is 3076 and that in 5 subsystems is 3212, which indicates increasing subsystems from 0 to 5 can reduce the complexity of MLPTS by 93.8% with increase of makespan by 4.4%. This demonstrates that HCCMS provides a way to significantly reduce the computational complexity in exchange for a small loss of optimization.

On the whole, this set of experiments demonstrates the obvious benefits of multiple subsystems to the resource management of the Cloud computing system. The experiments in the following subsections will also be carried out in the HCCMS structure.

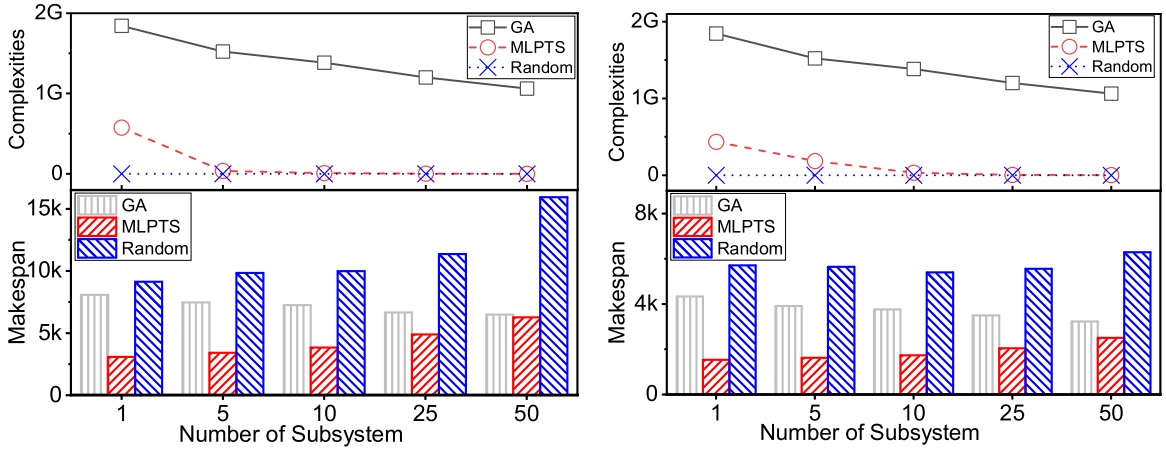
5.2.2. EX_1: Baseline strategies vs. DL-based selectors

In this set of experiments, we compare baseline strategies corresponding to Table 6 with DLSs corresponding to Table 4 to evaluate the optimality and adaptability of DLS in various scenarios.

We experiment in the simulated HCCMS with $q = 25$ subsystems, where numbers of each type of problems are all 5. The cost weights of subsystems are also in uniform distribution where $w_{\omega}^{(k)}(t) \sim$

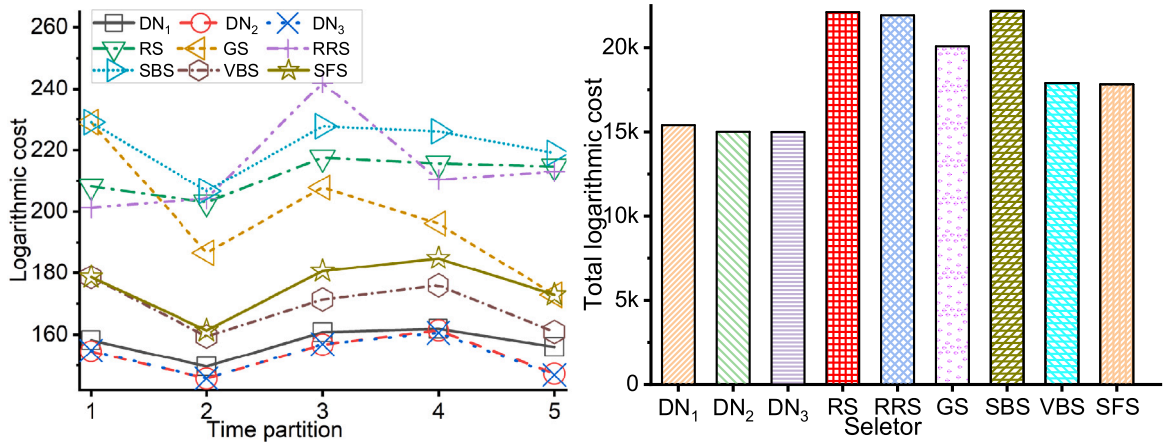
Table 7
Parameter setting in experiments.

Objective	Subsystem type	User type	Characteristic parameter	Parameter value	Constraint
$\min \omega_k^{(1)}$	Homogeneous full	Specific user groups	Processing time ET_{kji}	U(30, 120) min	$RT_{ki} \leq 1$ day
$\min \omega_k^{(2)}$	Heterogeneous full	Specific user groups	Processing time ET_{kji}	U(30, 120) min	$RT_{ki} \leq 1$ day
$\min \omega_k^{(3)}$	Homogeneous partial	Users requesting DS	Disk storage υ_{kji}^{DS}	U($10^3, 5 \times 10^4$) MB	$L_{ki}^{DS} \leq 10^5$ MB
$\min \omega_k^{(4)}$	Homogeneous partial	Users requesting CPU	CPU capacity υ_{kji}^{CPU}	U(10, 100) MIPS	$L_{ki}^{CPU} \leq 10^3$ MIPS
$\min \omega_k^{(5)}$	Heterogeneous partial	Users requesting CPU	CPU capacity υ_{kji}^{CPU}	U(10, 100) MIPS	$L_{ki}^{CPU} \leq 10^3$ MIPS



(a) 100 server nodes. (b) 200 server nodes.

Fig. 6. The makespan and computational complexities of various system organizations for 10 000 tasks.



(a) The cost per time partition for 5 time partitions. (b) The total cost for 100 time partitions.

Fig. 7. The performance comparison between baseline strategies and DL-based selectors with 25 subsystems for $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

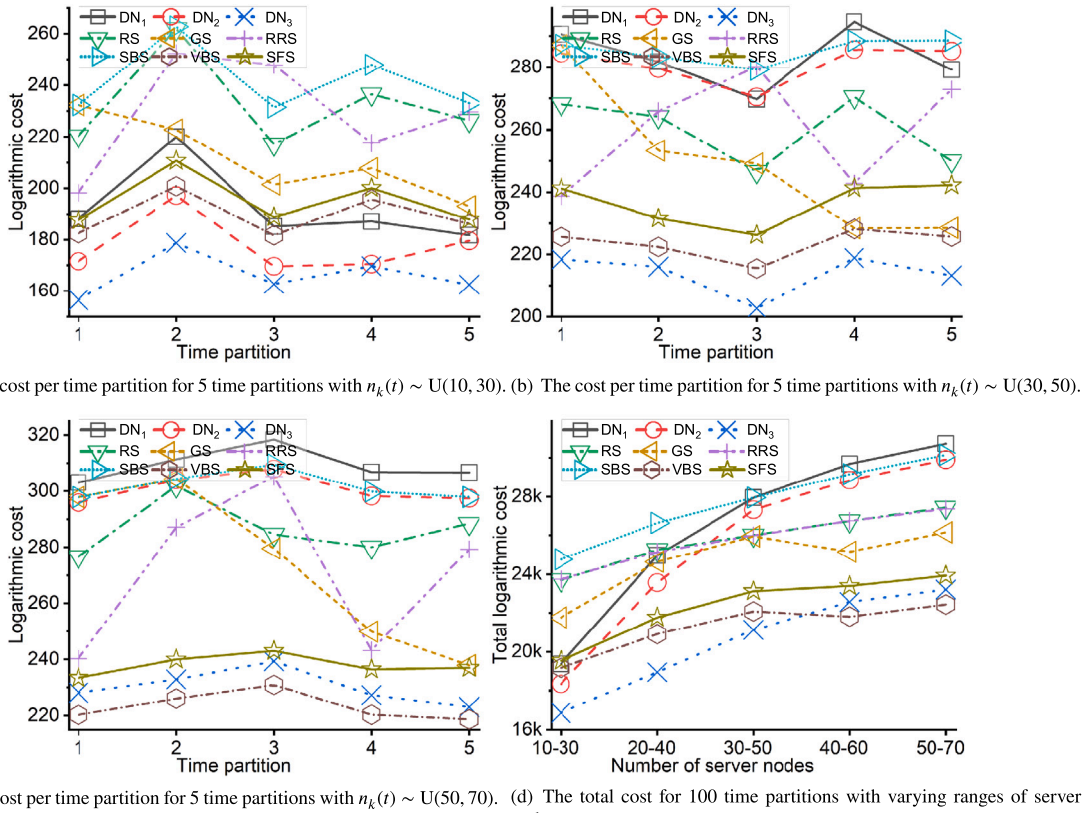
$\mathrm{U}(0, 100)/100$ and $w_\tau^{(k)}(t) \sim \mathrm{U}(0, 100)/100$ where $\mathrm{U}(0, 100)$ means randomly generating an integer between $[0, 100]$ subject to uniform distribution. As the management of HCCMS in this paper mainly concerns the cost of the system, thus we only use $\log U_k(t)$ (logarithmic cost) calculated by solution quality and complexity as Eq. (12) to evaluate the performance of selectors instead of additionally present optimization results in detail such as makespan and standard deviation for each subproblem. Dividing time into multi partitions, the number of the arriving tasks and the available resources in each time partition are generated by the uniform distribution. Additionally, the three DLSs, i.e. DN_1, DN_2 and DN_3 with different structures of networks, have been well trained in dataset for the distributions of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

We carry out experiments in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$: we randomly generate a set of tasks through these probability distributions, execute the selected algorithms using different algorithm selection strategies to schedule of generated tasks to meet the specified optimization objectives, record the execution time and optimization results of their selected scheduling algorithms, and then calculate the logarithmic cost according to the cost model as the index of the performance evaluation. Then, we plot the experiment results of logarithmic cost per time partition and total logarithmic cost in Fig. 7.

Fig. 7(a) plots the cost of the system at each time partition resulted from DLSs and the baseline strategies including random selector (denoted as RS in figures), greedy selector (GS), RR selector (RRS), single best selector (SBS), virtual best selector (VBS) and single fast selector (SFS). Each selector processes the same task under the same configuration of subsystems. In Fig. 7(a), the three DLSs that DN_1, DN_2 and DN_3 obtain less cost than baselines, where DN_3 performs best

Table 8
The comparison of total cost for 100 time partitions with 25 subsystems between DN_3 and baselines in the scenarios of $n_k(t) \sim U(2, 20)$ and $m_k(t) \sim U(n_k(t), 100)$.

Items	DN_3	RS	RRS	GS	SBS	VBS	SFS
Total cost	15 001	22 114	21 924	20 084	22 188	17 902	17 823
Improvement ($I_{baseline-DN_3}$)	–	7112	6923	5083	7187	2901	2822
$I_{baseline-DN_3}/DN_3$ (%)	–	47.4	46.1	33.9	47.9	19.3	18.8



(a) The cost per time partition for 5 time partitions with $n_k(t) \sim U(10, 30)$. (b) The cost per time partition for 5 time partitions with $n_k(t) \sim U(30, 50)$.

(c) The cost per time partition for 5 time partitions with $n_k(t) \sim U(50, 70)$. (d) The total cost for 100 time partitions with varying ranges of server nodes.

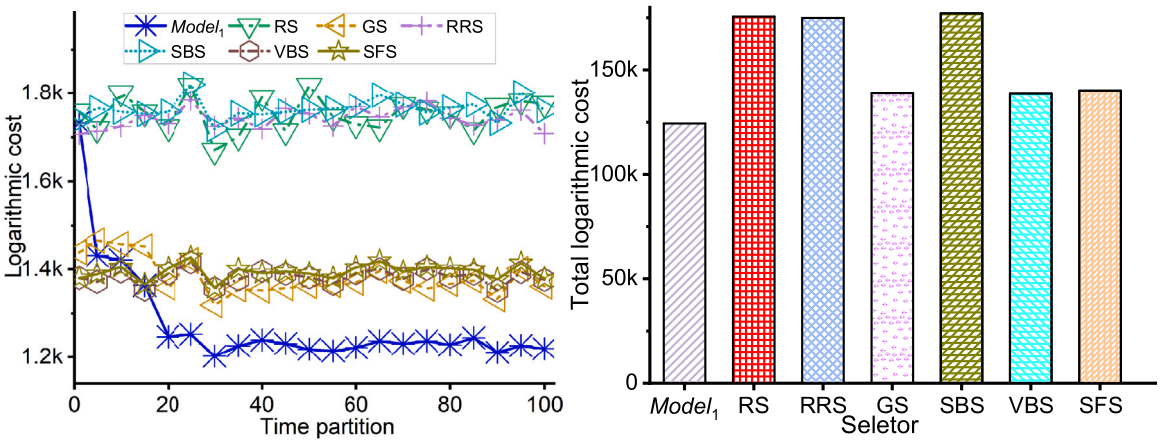
Fig. 8. The performance comparison between baseline strategies and DL-based selectors with 25 subsystems with varying ranges of server nodes and $m_k(t) \sim U(n_k(t), 100)$.

among all the DLSs. The results in Fig. 7(a) show that DLS can be used to reduce the cost of resource scheduling and increasing the number of neural network layers can improve the performance of algorithm selection. Fig. 7(b) plots the total cost for 100 time partitions where the tasks and subsystems for each selector are also the same. In terms of the total cost over a long time span, DLSs still outperform compared baseline strategies.

First of all, Fig. 7 shows that it is difficult to adapt to all scenarios with only one algorithm or some fixed algorithms, which proves once again the significance of selecting appropriate scheduling algorithms for different scenarios. Moreover, the performance of randomly or greedily selecting an algorithm to solve the resource scheduling problem is unstable. Then, Table 8 lists the total costs of Fig. 7(b) to observe the relative improvement of DN_3 compared with the baseline algorithm. Overall from Table 8, the total cost of DN_3 is improved by 47.4%, 46.1%, 33.9%, 47.9%, 19.3% and 18.8% respectively compared to RS, GS, RRS, SBS, VBS and SFS for 100 time partitions with 25 subsystems in the scenarios of $n_k(t) \sim U(2, 20)$ and $m_k(t) \sim U(n_k(t), 100)$.

In the experiments of Fig. 7 where DN_1, DN_2 and DN_3 significantly outperform baselines, the ranges of server nodes and tasks are that corresponding to the training dataset, i.e. $n_k(t) \sim U(2, 20)$ and $m_k(t) \sim U(n_k(t), 100)$. This implies DLS can perform well in the parameters they are repeatedly trained. However, models based on deep learning may not be adaptable to scenarios other than their training dataset. Considering this, we continue to evaluate the performance and validity of DLS for untrained parameter ranges. We changed the ranges of server nodes, retain the ranges of tasks as $m_k(t) \sim U(n_k(t), 100)$ and carried out several groups of experiments to observe the logarithmic cost of the DN_1, DN_2 and DN_3 which are trained in $n_k(t) \sim U(2, 20)$ and $m_k(t) \sim U(n_k(t), 100)$.

The results are plotted in Fig. 8, where Figs. 8(a), 8(b) and 8(c) plot the logarithmic cost per time partition for 5 time partitions respectively for the ranges $n_k(t) \sim U(10, 30)$, $n_k(t) \sim U(30, 50)$ and $n_k(t) \sim U(50, 70)$. Then, Fig. 8(d) plots the total cost for 100 time partitions for the ranges from $n_k(t) \sim U(10, 30)$ to $n_k(t) \sim U(50, 70)$. In Fig. 8(a) where $n_k(t) \sim U(10, 30)$ slightly deviates from the range of training $n_k(t) \sim U(2, 20)$, DN_1, DN_2 and DN_3 still outperform baselines. However, DN_1, DN_2 and DN_3 differ greater in performance than that of Fig. 7. DN_3 has significantly lower cost than DN_2 and DN_1, as well as DN_1 has close cost with VBS. In Fig. 8(b) where $n_k(t) \sim U(30, 50)$, the differences of cost between DN_1, DN_2 and DN_3 further increase, where DN_3 still outperform baselines but DN_1 and DN_2 achieve worse cost than other baselines except SBS. When in Fig. 8(c), performances of the three DLSs continue to decline so that VBS exceeds DN_3 although DN_3 still performs better than other baselines. Fig. 8(d) provides an overall trend of performance changing with the ranges of server nodes. In Fig. 8(d), the performance degradation speeds of DN_2 and DN_1 are faster than that of DN_3. When the range of server nodes reaches $U(20, 40)$, the costs of DN_1 and DN_2 become larger than that of VBS and SFS in baselines, as well as until that range reaches or exceeds $U(40, 60)$, the costs of DN_3 become greater than that of VBS and still less than SFS.



(a) The cost per time partition. (b) The total cost.

Fig. 9. The performance comparison between the DRLS ($Model_1$ being trained) and baseline strategies with 200 subsystems for 100 time partitions in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Table 9
The comparison of total cost for 100 time partitions with 200 subsystems between $Model_1$ and baselines in the scenarios $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Items	$Model_1$	RS	RRS	GS	SBS	VBS	SFS
Total cost	124 426	175 572	174 959	138 977	177 077	138 744	140 024
Improvement ($I_{baseline-Model_1}$)	–	51 146	50 533	14 551	52 651	14 318	15 598
$I_{baseline-Model_1}/Model_1$ (%)	–	41.1	40.6	11.7	42.3	11.5	12.5

Analyzing the overall results of experiments in Fig. 8 can gain that the performance of DLS, although with certain adaptability, will decline with the ranges of server nodes far away from that trained by DLS. This also means that DLS can maintain performance in static scenes, but cannot guarantee adaptability in dynamic scenes. Comparing DN_1, DN_2 and DN_3 in Figs. 8 and 7, we can see DN_3 is least affected by the range of server nodes in terms of performance followed by DN_2. In addition to the differences in network structure, these three DLSs have the same training strategy and dataset, as well as have approximate accuracies when achieving convergence. Therefore, this reveals a possible reason that a neural network-based DLS with more neurons or connection layers may have stronger adaptability in algorithm selection for varying ranges of parameters, consistent with that the better performing DN_3 has more layers and neurons than DN_2 and DN_1.

5.2.3. EX_2: Baseline strategies vs. $Model_1$

The premise of using DLS is enough historical data in advance to support training. From the above results of Figs. 7 and 8, after repeatedly training within the specified parameter range, DLS did learn the optimal selection within this parameter range, but its performance is still limited for the parameter range outside the training. Therefore, although DLS has better performance than the comparison strategies in static selection, we still need to explore additional selectors to tackle the dynamic selection of scheduling algorithms, which indicates the necessity of DRLS.

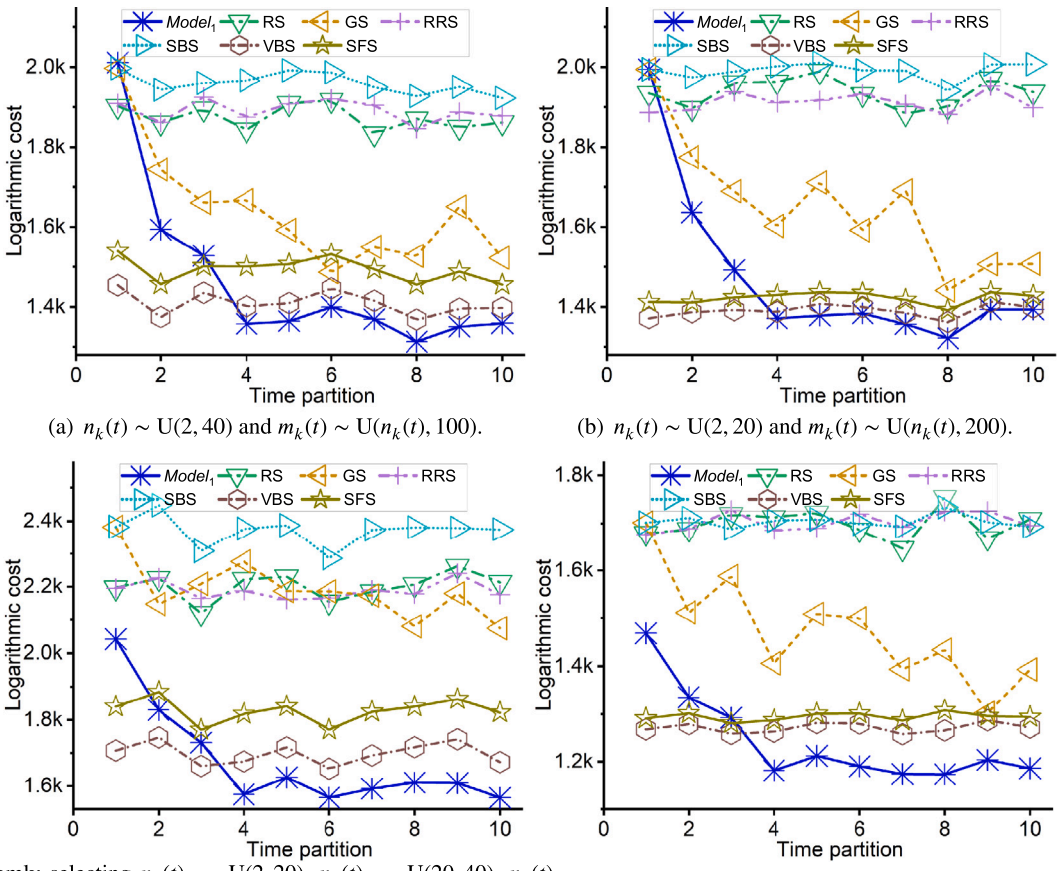
In this set of experiments, we compare baseline strategies corresponding to Table 6 with a DRLS ($Model_1$) to evaluate the optimality and adaptability of DRLS. We experiment in a simulated HCCMS with $q = 200$ subsystems, where numbers of each type of problems are all 40. The numbers of the arriving tasks and the available resources in each time partition are generated by uniform distribution or other distribution (in some experiments). The cost weights of subsystems are also in uniform distribution where $w_{\omega}^{(k)}(t) \sim \mathrm{U}(0, 100)/100$ and $w_{\tau}^{(k)}(t) \sim \mathrm{U}(0, 100)/100$. The above parameter settings are the same as those of the comparative experiments EX_1. The difference is that in the process of training DRLS, it is unknown which scheduling algorithm is the best in advance. DRLS finds a better selection through exploration and exploitation. Additionally, we choose $Model_1$ (without additional training strategies) as the instance of DRLS to observe the inherent advantages of DRL-based algorithm selector.

Since DRLS does not have the ability to optimize decision-making at the beginning before training which needs enough training to participate in decision-making, we set a long time period with 100 time partitions to observe the logarithmic cost and carry out experiments in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$. Then, the results of the logarithmic costs corresponding to the comparison baseline strategies and DRLS ($Model_1$) are plotted in Fig. 9.

Fig. 9(a) plots the cost of the system at each time partition. As shown in Fig. 9(a), before the 10-th time partition, the costs of $Model_1$ are not obviously smaller than the best baseline, which is because $Model_1$ selector has not been well trained. After the 10-th time partition, $Model_1$ can maintain better than comparison strategies with adequate training. This is because DRLS can learn the best selection strategy in the current scenario after enough time partitions. When time goes on, DRLs with lifelong learning will continue to evolve itself. Fig. 9(b) plots the total cost for 100 time partitions of the training process. In Fig. 9(b), $Model_1$ achieves the minimum total cost much less than baselines, although the cost of $Model_1$ is large in the initial few time partitions.

This set of experiments has validated the feasibility of DRLS for the scenarios without labeled data. Additionally, using DRLS can reduce the cost evidently for dynamic resource scheduling. Table 9 lists the total costs of Fig. 9(b) to observe the relative improvement of $Model_1$ compared with the baseline algorithm. Overall from Table 9, the total cost of $Model_1$ is improved by 41.1%, 40.6%, 11.7%, 42.3%, 11.5% and 12.5% respectively compared to RS, GS, RRS, SBS, VBS and SFS for 100 time partitions with 200 subsystems in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Similar to the experiments of DLS in Section 5.2.2, we carry out multi experiments to test the validity of DRLS in various ranges of parameters. We respectively change the distributions of server nodes and tasks, then plot the cost per time partition of $Model_1$ and baselines in the scenarios with 200 subsystems for 10 time partitions in Fig. 10. Comparing to Fig. 9, experiments in Fig. 10(a) change the range of server nodes to $n_k(t) \sim \mathrm{U}(2, 40)$; that in Fig. 10(b) change the range of tasks to $m_k(t) \sim \mathrm{U}(n_k(t), 200)$; that in Fig. 10(c) select $n_k(t) \sim \mathrm{U}(2, 20)$,



(a) $n_k(t) \sim \mathrm{U}(2, 40)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

(b) $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 200)$.

(c) Randomly selecting $n_k(t) \sim \mathrm{U}(2, 20)$, $n_k(t) \sim \mathrm{U}(20, 40)$, $n_k(t) \sim \mathrm{U}(30, 70)$, $n_k(t) \sim \mathrm{U}(50, 90)$ with equal probability in each time partition and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

(d) $n_k(t) \sim \mathrm{Tr}(2, 20)$ and $m_k(t) \sim \mathrm{Tr}(n_k(t), 100)$.

Fig. 10. The cost per time partition of $Model_1$ being trained and baseline strategies with 200 subsystems for 10 time partitions and various distributions of numbers of server nodes and tasks.

$n_k(t) \sim \mathrm{U}(20, 40)$, $n_k(t) \sim \mathrm{U}(30, 70)$, $n_k(t) \sim \mathrm{U}(50, 90)$ with equal probability; as well as that in Fig. 10(d) change the distribution of server nodes and tasks to triangular distributions i.e. $n_k(t) \sim \mathrm{Tr}(2, 20)$ and $m_k(t) \sim \mathrm{Tr}(n_k(t), 100)$ where the probability of $\mathrm{Tr}(a, b)$ is

$$P_{\mathrm{Tr}(a,b)}(x) = \frac{4 \min(|x-a|, |x-b|)}{(b-a)^2 - ((b-a) \mod 2)} \tag{18}$$

where $a \le x \le b \in \mathbb{N}^+$.

As shown in Fig. 10, DRLS can reach better performance than the baseline within 5 time partitions in various distributions of server nodes and tasks. Compared to that in Figs. 10(a) and 10(b), DRLS (i.e. $Model_1$) in Figs. 10(c) and 10(d) has greater advantages than baselines, which indicates that DRLS has stronger adaptability to more complex dynamic scenes than baselines. Differentiating from DLS, DRLS retains better performance than baselines with the change of parameters. This is because DRLS is continuously trained by the experience in each time partition, which makes DRLS adaptable to the dynamic varying scenarios. In addition, DRLS based on deep reinforcement learning not only learns the decision of algorithm selection, but also learns the properties of parameter distribution of tasks and server nodes to a certain extent, which is also helpful to improve the performance of DRLS in dynamic scenes.

5.2.4. EX_3: Comparison between $Model_1$ to $Model_5$

Previous experiments have verified the availability and superiority of DRLS. In this set of experiments, we further consider the strategies of DRLS including pre-trained model of DLS, LTPU and joint training corresponding to Table 5.

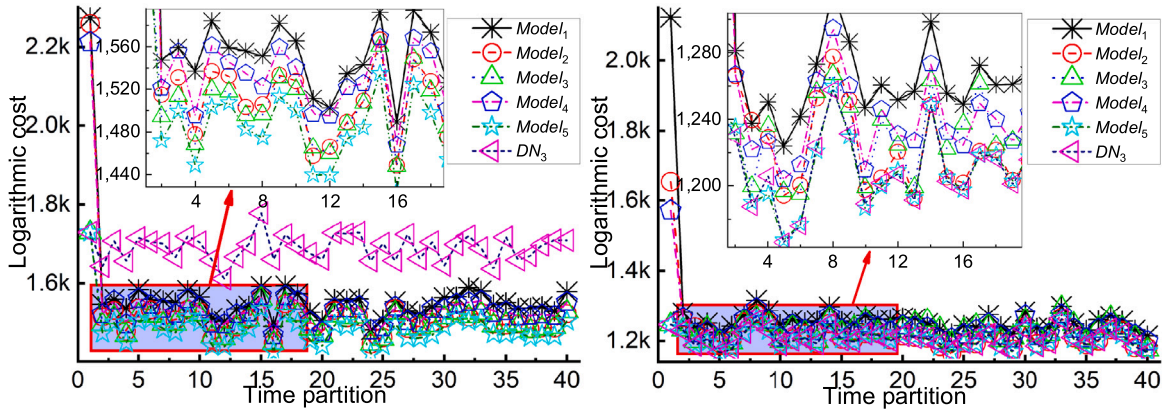
We carry out experiments in the simulated HCCMS with $q = 200$ subsystems. The cost weights of subsystems are also in uniform distribution where $w_\omega^{(k)}(t) \sim \mathrm{U}(0, 100)/100$ and $w_\tau^{(k)}(t) \sim \mathrm{U}(0, 100)/100$. The numbers of the arriving tasks and the available resources in each time partition are generated by the uniform distribution. In order to test the adaptability of DRLS for dynamic algorithms selection, we experiment with two groups of parameters generation:

- $n_k(t) \sim \mathrm{U}(2, 50)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 200)$.
- $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

The parameters generation of the first group is the same as the training dataset of DLS such as DN_3. Therefore, we also add the cost of DN_3 to participate in the comparison.

The pre-trained model being used in this set of experiments is the DN_3 in Section 5.2.2 which is trained under the parameters $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$. We observe the training process of $Model_1$ to $Model_5$ in 40 time partitions, and plot the results of the costs for each time partition in Fig. 11. Fig. 11(a) plots the results for the parameters generation of $n_k(t) \sim \mathrm{U}(2, 50)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 200)$, and Fig. 11(b) for that of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

From Fig. 11(a), $Model_5$, simultaneously leveraging pre-trained model, LTPU and joint training, has the fastest convergence speed and the lowest cost. Ranking from lowest to highest logarithmic cost gains $Model_5 < (Model_3, Model_2) < Model_4 < Model_1 < \mathrm{DN}_3$. $Model_2 < Model_4 < Model_1$ illustrates joint training can improve the optimization of decision-making with a small range, but not better than strategy of LTPU. $Model_3$, combining LTPU and pre-trained model, is better than $Model_2$ in the first 15 time partitions, but their results are very



(a) $n_k(t) \sim \mathrm{U}(2, 50), m_k(t) \sim \mathrm{U}(n_k(t), 200)$. (b) $n_k(t) \sim \mathrm{U}(2, 20), m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Fig. 11. The performance comparison in train process of the $Model_1$ to $Model_5$ with 200 subsystems for 40 time partitions.

close after 15 partitions. This demonstrates usage of pre-trained model can improve convergence speed however cannot evidently improve optimization of decision-making for the model after sufficient training. Additionally, the DN_3 without re-training has the higher costs than all of DRLS. This demonstrates DLS cannot adapt to dynamic selection of algorithms once the scenario is different from the trained dataset.

However in Fig. 11(b), DN_3 is close to $Model_5$ and outperforms than all of DRLS. The reason is that: during training DN_3 under the parameters of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$, the trained labels of dataset correspond to the best algorithms, i.e., DN_3 has learned the best selection; however in training of DRLS, the best algorithms are unknown and DRLS only learned a relatively high score selection through the exploration for better selections. Thus, the choose of the algorithms selectors depends on the scenarios.

In general, each strategy of pre-trained model, LTPU and joint training can improve the training or decision-making performance of the DRLS. And DRLS is more appropriate to resolve the dynamic selection in the real-time scheduling process when the parameters are unknown in advance. The $Model_5$ using all strategies simultaneously can achieve the best performance in the process of algorithm selection.

5.3. Overall summary

Through the multiple sets of experiments from different sights in this section, we can observe both DLS and DRLS outperform than baseline strategies in various resource scheduling scenarios of HCCMS. Among these experiments:

- EX_0 demonstrates the multi-layer system structure of HCCMS can greatly reduce the computational complexity of resource scheduling with little loss of optimization.
- EX_1 demonstrates DLS can reduce the whole cost significantly compared with baselines in the scenarios with stable parameter ranges where DN_3 reduces the cost by 47.4%, 46.1%, 33.9%, 47.9%, 19.3% and 18.8%, respectively compared to RS, GS, RRS, SBS, VBS and SFS. Additionally, EX_1 also demonstrates the performance of DLS will decline with the ranges of server nodes far away from that trained by DLS, as well as DLS with more layers or neurons may have stronger adaptability in algorithm selection for varying ranges of parameters.
- EX_2 demonstrates DRLS can obtain far better cost than baselines in the dynamic scheduling scenarios without labeled data where $Model_1$ reduces the cost by 41.1%, 40.6%, 11.7%, 42.3%, 11.5% and 12.5% respectively compared to RS, GS, RRS, SBS, VBS and SFS. Additionally, EX_2 also demonstrates DRLS retains better performance than baselines with the change of parameters.
- Finally, EX_3 demonstrates the effect of different strategies of DRLS where the simultaneous usage of DL-based pre-trained model, LTPU and joint training performs the best among all the DRLSs, as well as validates DRLS have stronger adaptability to dynamic scenes than DLS.

6. Conclusion and future works

In this paper, we formulate the joint scheduling problem of HCCMS combining five types of subproblems in four types of subsystems, which always cannot be addressed by a single scheduling algorithm. Focusing on this issue, we propose the scheduling framework to select the scheduling algorithms (SFSSA) to meet the resource management of complex HCCMS. To concretize SFSSA, we proposed DLS and DRLS, which can learn algorithm selection decisions in various scenarios to tackle the challenging joint scheduling problem of HCCMS. To improve the optimality and convergence of DRLS, we further apply various strategies including pre-trained model, long experience reply and joint training. Then, we carry out four sets of experiments to evaluate the performance of DLS and DRLS in the joint scheduling problem of reducing the cost of HCCMS.

From extensive experiments in multiple sights, we can conclude not only HCCMS structure can significantly improve the speed of resource management, but also our proposed SFSSA, with DLS and DRLS as instances both outperforming baselines, can address the joint scheduling problem in HCCMS. Concretely, DLS reduced the cost by 47.4%, 46.1%, 33.9%, 47.9%, 19.3% and 18.8% in the scenarios with stable parameter ranges; DRLS reduced the cost by 41.1%, 40.6%, 11.7%, 42.3%, 11.5% and 12.5% in the dynamic scenarios compared to RS, GS, RRS, SBS, VBS and SFS respectively. We can also conclude that DRLS has stronger adaptability to dynamic resource scheduling scenarios than DLS. Additionally, the strategies, i.e. pre-trained model, long-term experience replay & period update and joint training, are all conducive to improving the performance of DRLS.

The obvious significance of DLS and DRLS is that they not only demonstrate the great potential of DL and DRL as algorithms selectors, but also prove that *resource scheduling algorithms can also be regarded as the resources to be scheduled*. Compared with directly using DL and DRL as scheduling algorithms, DLS and DRLS (using DL and DRL as algorithm selectors) have much lower computational complexity, which also provides more possibilities for their application in realistic complex systems. It also puts forward a novel considerable solution to the challenge that *no scheduling algorithm is suitable for all scenarios*, where SFSSA with DLS and DRLS implies that *since there is no such algorithm suitable for all scenarios, we will choose the most appropriate algorithm for different scenarios*. In SFSSA, all scheduling algorithms are regarded as useful "treasures", because we could always find a certain

scenario for each algorithm, making this algorithm superior to all other algorithms in some aspects. On other counts, this paper also illustrates the potential of hierarchical management of Cloud computing systems using multiple subsystems.

In the future work, we plan to explore more structures and strategies of DLS and DRLS to adapt to more complex resource scheduling scenarios, as well as we plan to continue to explore the dynamic configuration and access control policy of resources and server nodes of HCCMS where we consider that the same server node can actually belong to different subsystems at different time.

CRediT authorship contribution statement

Guangyao Zhou: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization, Funding acquisition. **Ruiming Wen:** Investigation, Writing – review & editing. **Wenhong Tian:** Resources, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Rajkumar Buyya:** Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was partially supported by National Key Research and Development Program of China [grant number 2018AAA0103203]; Project of Sichuan Provincial Department of science and technology [grant number 2021YFG0325]; and National Natural Science Foundation of China [grant number 61672136, 61828202].

References

Abhikriti, N., Sunita, D., 2017. Enhanced task scheduling algorithm using multi-objective function for cloud computing framework. In: International Conference on Next Generation Computing Technologies. pp. 110–121.

Adhikari, M., Amgoth, T., Srirama, S.N., 2019. A survey on scheduling strategies for workflows in cloud environment and emerging trends. ACM Comput. Surv. 52 (4), 68:1–68:36. http://dx.doi.org/10.1145/3325097.

Al-Mahruqi, A.A.H., Morison, G., Stewart, B.G., Vallavaraj, A., 2021. Hybrid heuristic algorithm for better energy optimization and resource utilization in cloud computing. Wirel. Pers. Commun. 118 (1), 43–73. http://dx.doi.org/10.1016/j.future.2019.08.012, URL: https://www.sciencedirect.com/science/article/pii/S0167739X19306983.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M., 2010. A view of cloud computing. Commun. ACM 53 (4), 50–58. http://dx.doi.org/10.1145/1721654.1721672.

Boas, M.G.V., Santos, H.G., de Campos Merschmann, L.H., Berghe, G.V., 2021. Optimal decision trees for the algorithm selection problem: integer programming based approaches. Int. Trans. Oper. Res. 28 (5), 2759–2781. http://dx.doi.org/10.1111/itor.12724.

Croce, F.D., Scatamacchia, R., 2020. The longest processing time rule for identical parallel machines revisited. J. Sched. 23 (2), 163–176. http://dx.doi.org/10.1007/s10951-018-0597-6.

Czako, Z., Sebestyen, G., Hangan, A., 2021. AutomaticaI - A hybrid approach for automatic artificial intelligence algorithm selection and hyperparameter tuning. Expert Syst. Appl. 182, 115225. http://dx.doi.org/10.1016/j.eswa.2021.115225, URL: https://www.sciencedirect.com/science/article/pii/S0957417421006576.

Deshpande, N., Sharma, N., Yu, Q., Krutz, D.E., 2021. R-CASS: using algorithm selection for self-adaptive service oriented systems. In: 2021 IEEE International Conference on Web Services, ICWS 2021, Chicago, IL, USA, September 5-10, 2021. IEEE, pp. 61–72. http://dx.doi.org/10.1109/ICWS53863.2021.00021.

Ding, D., Fan, X., Zhao, Y., Kang, K., Yin, Q., Zeng, J., 2020. Q-learning based dynamic task scheduling for energy-efficient cloud computing. Future Gener. Comput. Syst. 108, 361–371. http://dx.doi.org/10.1016/j.future.2020.02.018, URL: https://www.sciencedirect.com/science/article/pii/S0167739X19313858.

Dong, T., Xue, F., Xiao, C., Li, J., 2020. Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. Concurr. Comput. Pract. Exp. 32 (11), http://dx.doi.org/10.1002/cpe.5654, URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5654.

Duc, T.L., Leiva, R.A.G., Casari, P., Östberg, P., 2019. Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. ACM Comput. Surv. 52 (5), 94:1–94:39. http://dx.doi.org/10.1145/3341145.

Fiandrino, C., Kliazovich, D., Bouvry, P., Zomaya, A.Y., 2017. Performance and energy efficiency metrics for communication systems of cloud computing data centers. IEEE Trans. Cloud Comput. 5 (4), 738–750. http://dx.doi.org/10.1109/TCC.2015.2424892.

Ghalami, L., Grosu, D., 2019. Scheduling parallel identical machines to minimize makespan: A parallel approximation algorithm. J. Parallel Distrib. Comput. 133, 221–231. http://dx.doi.org/10.1016/j.jpdc.2018.05.008, URL: https://www.sciencedirect.com/science/article/pii/S0743731518303691.

Guan, Z., Melodia, T., 2017. The value of cooperation: Minimizing user costs in multi-broker mobile cloud computing networks. IEEE Trans. Cloud Comput. 5 (4), 780–791. http://dx.doi.org/10.1109/TCC.2015.2440257.

Gudu, D., Hardt, M., Streit, A., 2018. Combinatorial auction algorithm selection for cloud resource allocation using machine learning. In: Euro-Par 2018: Parallel Processing - 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings. In: Lecture Notes in Computer Science, vol. 11014, Springer, pp. 378–391. http://dx.doi.org/10.1007/978-3-319-96983-1_27.

Guo, W., Tian, W., Ye, Y., Xu, L., Wu, K., 2021. Cloud resource scheduling with deep reinforcement learning and imitation learning. IEEE Internet Things J. 8 (5), 3576–3586. http://dx.doi.org/10.1109/JIOT.2020.3025015.

Hong, Z., Chen, W., Huang, H., Guo, S., Zheng, Z., 2019. Multi-hop cooperative computation offloading for industrial IoT-edge-cloud computing environments. IEEE Trans. Parallel Distrib. Syst. 30 (12), 2759–2774. http://dx.doi.org/10.1109/TPDS.2019.2926979.

Hu, H., Li, Z., Hu, H., Chen, J., Ge, J., Li, C., Chang, V.I., 2018. Multi-objective scheduling for scientific workflow in multicloud environment. J. Netw. Comput. Appl. 114, 108–122. http://dx.doi.org/10.1016/j.jnca.2018.03.028, URL: https://www.sciencedirect.com/science/article/pii/S1084804518301152.

Huerta, I.I., Neira, D.A., Ortega, D.A., Varas, V., Godoy, J., Achá, R.J.A., 2022. Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection. Expert Syst. Appl. 187, 115948. http://dx.doi.org/10.1016/j.eswa.2021.115948, URL: https://www.sciencedirect.com/science/article/pii/S0957417421013014.

Iranmanesh, A., Naji, H.R., 2021. DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing. Clust. Comput. 24 (2), 667–681. http://dx.doi.org/10.1007/s10586-020-03145-8.

Ismayilov, G., Topcuoglu, H.R., 2020. Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing. Future Gener. Comput. Syst. 102, 307–322. http://dx.doi.org/10.1016/j.future.2019.08.012, URL: https://www.sciencedirect.com/science/article/pii/S0167739X19306983.

Kardani-Moghaddam, S., Buyya, R., Ramamohanarao, K., 2021. ADRL: a hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds. IEEE Trans. Parallel Distrib. Syst. 32 (3), 514–526. http://dx.doi.org/10.1109/TPDS.2020.3025914.

Karthiban, K., Raj, J.S., 2020. An efficient green computing fair resource allocation in cloud computing using modified deep reinforcement learning algorithm. Soft Comput. 24 (19), 14933–14942. http://dx.doi.org/10.1007/s00500-020-04846-3.

Kumar, M., Sharma, S.C., Goel, A., Singh, S.P., 2019. A comprehensive survey for scheduling techniques in cloud computing. J. Netw. Comput. Appl. 143, 1–33. http://dx.doi.org/10.1016/j.jnca.2019.06.006, URL: https://www.sciencedirect.com/science/article/pii/S1084804519302036.

Laili, Y., Lin, S., Tang, D., 2020. Multi-phase integrated scheduling of hybrid tasks in cloud manufacturing environment. Robot. Comput. Integr. Manuf. 61, 101850. http://dx.doi.org/10.1016/j.rcim.2019.101850, URL: https://www.sciencedirect.com/science/article/pii/S0736584519301188.

Li, L., 2009. An optimistic differentiated service job scheduling system for cloud computing service users and providers. In: 2009 Third International Conference on Multimedia and Ubiquitous Engineering, MUE 2009, Qingdao, China, June 4-6, 2009. IEEE Computer Society, pp. 295–299. http://dx.doi.org/10.1109/MUE.2009.58.

Li, J., 2020. Resource optimization scheduling and allocation for hierarchical distributed cloud service system in smart city. Future Gener. Comput. Syst. 107, 247–256. http://dx.doi.org/10.1016/j.future.2019.12.040, URL: https://www.sciencedirect.com/science/article/pii/S0167739X1932028X.

Li, M., Yu, F.R., Si, P., Wu, W., Zhang, Y., 2020. Resource optimization for delay-tolerant data in blockchain-enabled IoT with edge computing: A deep reinforcement learning approach. IEEE Internet Things J. 7 (10), 9399–9412. http://dx.doi.org/10.1109/JIOT.2020.3007869.

Liu, J., Xiong, K., Ng, D.W.K., Fan, P., Zhong, Z., Letaief, K.B., 2020. Max-min energy balance in wireless-powered hierarchical fog-cloud computing networks. IEEE Trans. Wirel. Commun. 19 (11), 7064–7080. http://dx.doi.org/10.1109/TWC.2020.3007805.

Liu, X.F., Zhan, Z., Deng, J.D., Li, Y., Gu, T., Zhang, J., 2018. An energy efficient ant colony system for virtual machine placement in cloud computing. IEEE Trans. Evol. Comput. 22 (1), 113–128. http://dx.doi.org/10.1109/TEVC.2016.2623803.

Lolos, K., Konstantinou, I., Kantere, V., Koziris, N., 2017. Elastic management of cloud applications using adaptive reinforcement learning. In: 2017 IEEE International Conference on Big Data (IEEE BigData 2017), Boston, MA, USA, December 11-14, 2017. IEEE Computer Society, pp. 203–212. http://dx.doi.org/10.1109/BigData.2017.8257928.

Lu, H., Gu, C., Luo, F., Ding, W., Liu, X., 2020. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. Future Gener. Comput. Syst. 102, 847–861. http://dx.doi.org/10.1016/j.future.2019.07.019, URL: https://www.sciencedirect.com/science/article/pii/S0167739X19308209.

Mahil, M., Jayasree, T., 2021. Combined particle swarm optimization and ant colony system for energy efficient cloud data centers. Concurr. Comput. Pract. Exp. 33 (10), http://dx.doi.org/10.1002/cpe.6195, URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6195.

Mao, J., Pan, Q., Miao, Z., Gao, L., 2021. An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. Expert Syst. Appl. 169, 114495. http://dx.doi.org/10.1016/j.eswa.2020.114495, URL: https://www.sciencedirect.com/science/article/pii/S0957417420311398.

Miao, Y., Wu, G., Li, M., Ghoneim, A., Al-Rakhami, M., Hossain, M.S., 2020. Intelligent task prediction and computation offloading based on mobile-edge cloud computing. Future Gener. Comput. Syst. 102, 925–931. http://dx.doi.org/10.1016/j.future.2019.09.035, URL: https://www.sciencedirect.com/science/article/pii/S0167739X19320862.

Muñoz, M.A., Kirley, M., 2021. Sampling effects on algorithm selection for continuous black-box optimization. Algorithms 14 (1), 19. http://dx.doi.org/10.3390/a14010019, URL: https://www.mdpi.com/1999-4893/14/1/19.

Nguyen, T.T., Ha, V.N., Le, L.B., Schober, R., 2020. Joint data compression and computation offloading in hierarchical fog-cloud systems. IEEE Trans. Wirel. Commun. 19 (1), 293–309. http://dx.doi.org/10.1109/TWC.2019.2944165.

Pradhan, Pandaba, Behera, Ku, P., Ray, B., 2016. Modified round robin algorithm for resource allocation in cloud computing. Procedia Comput. Sci. 85, 878–890. http://dx.doi.org/10.1016/j.procs.2016.05.278, URL: https://www.sciencedirect.com/science/article/pii/S1877050916306287.

Seiler, M., Pohl, J., Bossek, J., Kerschke, P., Trautmann, H., 2020. Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem. In: Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, the Netherlands, September 5-9, 2020, Proceedings, Part I. In: Lecture Notes in Computer Science, vol. 12269, Springer, pp. 48–64. http://dx.doi.org/10.1007/978-3-030-58112-1_4.

Sofia, A.S., Ganeshkumar, P., 2018. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using NSGA-II. J. Netw. Syst. Manage. 26 (2), 463–485. http://dx.doi.org/10.1016/j.matpr.2020.11.556, URL: https://www.sciencedirect.com/science/article/pii/S2214785320392257.

Sudarshan Chakravarthy, A., Sudhakar, C., Ramesh, T., 2019. Energy efficient VM scheduling and routing in multi-tenant cloud data center. Sustain. Comput.: Inform. Syst. 22, 139–151. http://dx.doi.org/10.1016/j.suscom.2019.04.004, URL: https://www.sciencedirect.com/science/article/pii/S2210537918303160.

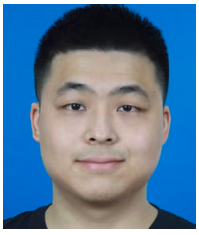
Tian, W., He, M., Guo, W., Huang, W., Shi, X., Shang, M., Toosi, A.N., Buyya, R., 2018. On minimizing total energy consumption in the scheduling of virtual machine reservations. J. Netw. Comput. Appl. 113, 64–74. http://dx.doi.org/10.1016/j.jnca.2018.03.033, URL: https://www.sciencedirect.com/science/article/pii/S1084804518301267.

Tong, Z., Chen, H., Deng, X., Li, K., Li, K., 2020. A scheduling scheme in the cloud computing environment using deep q-learning. Inform. Sci. 512, 1170–1191. http://dx.doi.org/10.1016/j.ins.2019.10.035, URL: https://www.sciencedirect.com/science/article/pii/S0020025519309971.

Wan, B., Dang, J., Li, Z., Gong, H., Zhang, F., Oh, S., 2020. Modeling analysis and cost-performance ratio optimization of virtual machine scheduling in cloud computing. IEEE Trans. Parallel Distrib. Syst. 31 (7), 1518–1532. http://dx.doi.org/10.1109/TPDS.2020.2968913.

Zhou, X., Zhang, G., Sun, J., Zhou, J., Wei, T., Hu, S., 2019. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. Future Gener. Comput. Syst. 93, 278–289. http://dx.doi.org/10.1016/j.future.2018.10.046, URL: https://www.sciencedirect.com/science/article/pii/S0167739X18314080.

Guangyao Zhou received Bachelor's degree and Master's degree from School of architectural engineering, Tianjin University, China. He is now a Ph.D candidate at School of information and software engineering, University of Electronic Science and Technology of China. His research interests include scheduling algorithms in Cloud Computing, facial expression recognition, algorithmic theory of machine learning and BigData processing.



Ruiming Wen is now a Ph.D candidate at the School of information and software engineering, University of Electronic Science and Technology of China. His main research interests include algorithmic theory of machine learning, facial expression recognition by deep learning, low illumination image processing, BigData processing and Cloud Computing.

Wenhong Tian received a Ph.D. degree from the Department of Computer Science, North Carolina State University, Raleigh, NC, USA. He is now a professor at the University of Electronic Science and Technology of China (UESTC). His research interests include scheduling in Cloud Computing and Bigdata platforms, and image recognition by deep learning. He has more than 110 journal/conference publications and 5 books in related areas.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He served as a Future Fellow of the Australian Research Council during 20122016. He received the Ph.D. degree in Computer Science and Software Engineering from Monash University, Melbourne, Australia, in 2002. He has authored over 750 publications and seven text books. He is one of the highly cited authors in computer science and software engineering worldwide (hindex=154, gindex=331, 125200+ citations). He is recognized as a "Web of Science Highly Cited Researcher" for six consecutive years since 2016, and Scopus Researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier for his outstanding contributions to Cloud Computing and distributed systems.