Deep Reinforcement Learning Enhanced Greedy Optimization for Online Scheduling of Batched Tasks in Cloud HPC Systems

Yuanhao Yang, Hong Shen

Abstract—In a large cloud data center HPC system, a critical problem is how to allocate the submitted tasks to heterogeneous servers that will achieve the goal of maximizing the system's gain defined as the value of completed tasks minus system operation costs. We consider this problem in the online setting that tasks arrive in batches and propose a novel deep reinforcement learning (DRL) enhanced greedy optimization algorithm of two-stage scheduling interacting task sequencing and task allocation. For task sequencing, we deploy a DRL module to predict the best allocation sequence for each arriving batch of tasks based on the knowledge (allocation strategies) learnt from previous batches. For task allocation, we propose a greedy strategy that allocates tasks to servers one by one online following the allocation sequence to maximize the total gain increase. We show that our greedy strategy has a performance guarantee of competitive ratio $\frac{1}{1+\kappa}$ to the optimal offline solution, which improves the existing result for the same problem, where κ is upper bounded by the maximum cost-to-gain ratio of each task. While our DRL module enhances the greedy algorithm by providing the likely-optimal allocation sequence for each batch of arriving tasks, our greedy strategy bounds DRL's prediction error within a proven worst-case performance guarantee for any allocation sequence. It enables a better solution quality than that obtainable from both DRL and greedy optimization alone. Extensive experiment evaluation results in both simulation and real application environments demonstrate the effectiveness and efficiency of our proposed algorithm. Compared with the state-of-the-art baselines, our algorithm increases the system gain by about 10% to 30%. Our algorithm provides an interesting example of combining machine learning (ML) and greedy optimization techniques to improve ML-based solutions with a worst-case performance guarantee for solving hard optimization problems.

Index Terms—task scheduling, deep reinforcement learning, greedy optimization, approximation algorithm

1 Introduction

Cloud computing with large-scale data centers brings remarkable high-performance computing (HPC) efficiency and cost-savings to the end-users. With the growing computation overheads and the scale of data centers, scheduling computation tasks from users on heterogeneous servers to maximize the system gain is critical. The problem in its simplest form of equal-value tasks (of different sizes) is known as NP-hard because it can be reduced to the classical bin-backing problem, letting alone that in the general case of tasks with different values. This paper addresses the problem of how to maximize the system gain and hence increase the revenue of an HPC cloud data center by increasing the total value of completed tasks and reducing the operation costs in the way of "pay-as-you-go" billing in cloud computing.

Many applications such as weather forecasting and target surveillance come in with a requirement on completion time [1], which can be either hard (before a deadline) or soft (as soon as possible). Their value (reward payable to HPC) is inversely proportional to their completion time. Therefore, task scheduling should take into account this practical need from HPC clients and prioritize applications of high value for execution. On the other hand, from HPC system vendors' point of view, for a given batch of applications, scheduling should allocate the fewest possible servers for their execution to reduce the operation costs (server launching, task execution and etc). Hence, combining these two factors defines our goal of maximizing system's gain to be that of maximizing the total value of completed applications minus system operation costs. Apparently, achieving this goal will enable an HPC cloud data center to increase its revenue continuously.

Task scheduling is a cruical component for attaining high performance of cloud data centers. In the literature, a number of task schedulers have been implemented and deployed by academia and industry. Authors in [1] presented an algorithm with $\frac{1}{2}$ approximation ratio to maximize total accrued utility value in discrete-time domain under the assumption that all servers are homogeneous and each server can only accept tasks. To handle the complex cloud environment without the above assumption, i.e., servers are heterogeneous and may run multiple tasks at the same time, an offline greedy basedscheduling algorithm was proposed in [2]. For commercial clouds, task scheduling is expected to achieve the goal of maximizing system gain measured by the value (reward) of task completion minus associated operation costs, including the launching cost of servers [3], execution cost determined primarily by energy consumption [4], and renting cost if the servers are rented from a third-party [5]. This problem offers great challenges as addressing either value or costs alone is already NP-hard from its reduction from the bin-packing and knapsack problems, letting alone considering them together because they often conflict with each other (e.g., reducing completion time implies increasing computing power allocation). Lots

• *Authors are with School of Computer Science and Engineering, Sun Yat-sen University, China. Corresponding author is Hong Shen. E-mail: shenh3@mail.sysu.edu.cn.*

of studies on designing a cost effective scheduler for the above goal have been done. For example, the scheduling algorithm proposed in [6] minimizes the task completion time under the given budget constraints, the work [5] presents an online algorithm with competitive ratio $\frac{\rho-2}{\delta}$ to maximize the system gain that considers only the price of buying servers, and [7] presents a genetic algorithm to optimize the resource allocation problem that considers multiple types of costs but without a guaranteed convergence of running time. Recently, deep reinforcement learning (DRL) has experienced tremendous growth and has been employed to successfully solve lots of NP-hard problems in cloud computing, like minimizing the makespan[8] and maximizing the server utilization rate [9]. Compared with traditional heuristic algorithms in clouds[10], DRL is able to explore an optimal policy via the interactions with true environments based on historical knowledge. However, the training stability and scalability of such a deep learning approach continue to be a critical problem. Particularly, the common bottleneck of no performance guarantee for all machine learning approaches obstacles their practical adoptability [11].

Our work in this paper is motivated by many practical delay-sensitive applications that not only enable learning via interactions with real environments which greedy optimization algorithms fail to cope with, but also require a worst-case performance guarantee which machine learning techniques are unable to offer. Examples of such applications in scalable areas include threat detection in air defense systems, radar tracking, target surveillance and tracking [1]. To overcome the above shortcomings in the existing work, we propose in this paper a novel approach of combining machine learning and greedy algorithm to maximize system gain in online scheduling of delay-sensitive tasks over heterogeneous servers in a cloud data center. Particularly, we propose a two-stage design of the task scheduler that ensembles a reinforcement learning based task allocation sequence generator and a greedy based gain maximization task allocator. For each arriving batch of tasks, our scheduler first decides the allocation order of these tasks applying reinforcement learning based on the historical data of scheduling information and then distribute the tasks to the available servers one by one following this order using a greedy approach of marginal gain maximization. With a proven competitive ratio of $\frac{1}{1+\kappa}$, our scheduler provides the worst-case performance guarantee from the optimal offline scheduling, where κ is upper bounded by the maximum cost-to-gain ratio of each task.

We summarize our contributions as follows.

- We propose a novel scheme of task sequencing and task allocation two-stage online scheduling on heterogeneous cloud servers by joint deep reinforcement learning and greedy optimization for online task scheduling. It combines the merits of strong prediction power of DRL and performance guarantee of greedy optimization.
- We develop an effective model of deep reinforcement learning to predict the allocation sequence for an online batch of arriving tasks based on the historical scheduling information of task sequences.
- We design an efficient greedy algorithm for allocating tasks to servers online following a given allocation sequence and show that the algorithm has a performance guarantee of competitive ratio $\frac{1}{1+\kappa}$ to the optimal offline task scheduling scheme for the objective of maximum system gain, which improves the known result for the same problem [5].
- We conduct extensive experiments to validate our proposed algorithm and show that its performance is remarkably superior to that of the state-of-the-art baselines.

The remainder of the paper is organized as follows: Section 2 discusses the related work. Section 3 mathematically formulates the problem. Section 4 outlines the framework of our two stage scheduling. Section 5 and Section 6 present the the scheduling algorithm and the corresponding analysis respectively. Section 7 presents the experimental results and performance comparison with the state-of-the-art baseline algorithms in both simulated and real application environments with real-world data. Section 8 concludes our work.

2 Related Work

The task scheduling problem for gain maximization has been widely studied in the literature. As this problem is NP-hard in general, optimal solutions can be found only for offline scheduling on a single server [12] and in special cases. Existing work can be roughly classified into two main categories of approximation algorithms and heuristic algorithms.

In the first category of approximation algorithms, for offline scheduling, Chen et al. [13] present a linear programming based approach for minmizing the completion time of MapReduce applications on homogeneous processors, achieving $\frac{1}{8}$ approximation ratio. Under the assumption that different tasks have different sensitivities to completion time, Li et al. recently propose an offline algorithm of $\frac{1}{2}$ approximation ratio that greedily makes task scheduling decisions by calculating the self-defined spatial and temporal interference [1] on homogeneous multi-processors. For the case of heterogeneous processors, the work [14] proposes an algorithm of $O(K)$ asymptotic average-case performance to minimize the overall completion time of parallel applications, where K is determined by the largest task size and processor number. However, the aforementioned work does not consider resource packing in real servers. To mitigate this issue, Cohen et al.[2] present a greedy based approach to minmize the weighted completion time on a practical system with heterogeneous servers, achieving an apprroximation ratio bounded by the ratio of highest resource demand of any job to the server's capacity. Different from these offline algorithms, when tasks arrive online, Xu et al. [15] propose an algorithm for scheduling parallel tasks and allocating resources in a cluster of multiple heterogeneous servers, with a competitive ratio related to the longest processing time of tasks and the resource capacity of the system. Allowing rejection of tasks when necessary, the work [16] proposes an ϵ^3 competitive ratio algorithm for non-preemptive scheduling, where ϵ is the fraction of rejected tasks. All these algorithms focus on the value that can be earned, but ignore the system operation cost. Typically, minimizing the total cost of used servers can be reduced from the online bin packing problem,

whose competitive ratio has a lower bound of max/min job duration ratio [17]. For consideration of both value and cost, the work [5] proposes an online algorithm with competitive ratio related to the cost to value ratio of all tasks to maximize the system gain. However, it neglects the system operation cost for task execution and hence is still impractical. Furthermore, it does not consider the situation that tasks may arrive in batches instead of one by one. In summary, approximation algorithms can ensure the worst-case performance guarantee, but are unable to learn from real environments to improve performance based on past knowledge.

In the other category, various heuristic approaches are used to achieve gain maximization. To reduce the completion time, Dietze et al.[18] design a search based heuristic policy to assign parallel tasks to heterogeneous servers. Moreover, considering that different jobs have different degrees of sensitivity to their completion time, the work [19] proposes a lexicographical optimization based heuristic approach to maximize the value system receives. Approaches in [20] and [21] propose to save cost and improve resource utilization via decreasing the number of used processors and servers respectively. The work [22] proposes to eliminate the energy cost by reducing the time interval between starting the first task and finishing the last task on the same server. Taking into account both value and cost, Gao et al. combine the genetic algorithm and the multi-agent optimization algorithm to maximize the system gain [7] and Yu et al. [6] also propose a genetic algorithm to schedule tasks under a specific budget constraint. Nevertheless, because of the long convergence time, they fail to be responsive in real production environments. Fortunately, recent breakthroughs in artificial intelligence provide a promising approach DRL, which can be trained offline on historical data and perform responsive actions online. For example, Mondal et al. exploit DRL to learn temporal resource usage patterns of time varying workloads and schedule tasks across heterogeneous machines [9]. DRL has demonstrated promising potential in many fields, but the performance stability and scalability are challenging to achieve, which obstacles its practical use [11]. In summary, heuristic algorithms are able to learn adaptively via interactions with dynamic environments and predict actions toward the optimization goal based on the learnt knowledge, but they fail to provide a worst-case performance guarantee.

To address the above shortcomings, this paper proposes a DRL enhanced greedy optimization for online task schedling, which combines the merits of strong learning power of DRL and performance guarantee of greedy optimization.

3 System Model and Problem Formulation

This section introduces our system model, then gives the problem formulation and shows its hardness.

3.1 System Model

We are given a cloud data center with heterogeneous servers and dynamically arriving batches of tasks, and are required to schedule each batch of tasks on their arrival for execution in such a way that can maximize the system's total gain.

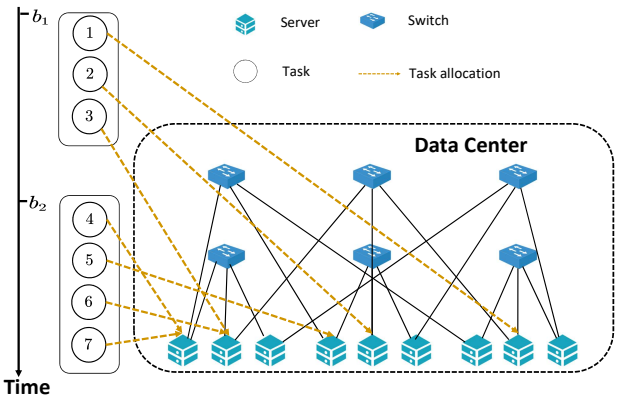


Fig. 1: The main process for cloud providers to schedule the submitted tasks

Fig. 1 depicts the system model and we describe each component below.

The data center is composed of a large number of heterogeneous servers interconnected via a high-speed interconnection network. Each server has its own capability of resources including CPUs of different clock speeds and memory.

Tasks are submitted from geographically distributed clients dynamically in the cloud environment. They form batches of arbitrary sizes following their arriving order, and each batch contains all the tasks that arrive simultaneously. The task is characterized by multiple attributes including arrival time, deadline for completion, workload, required resources and value.

Following the same assumptions as [5], for simplicity we assume:

- Each task has a value which is inversely proportional to its completion time before the deadline.
- Starting up a server (by the first task allocated to it) will generate a server launching cost proportional to the server's capacity.
- Executing a task will result in an execution cost proportional to the execution length.

So our scheduling(dashed lines in Fig. 1) is to find an allocation of tasks to servers that maximizes the total system gain defined as value minus costs.

3.2 Problem Formulation

Given a set of heterogeneous servers with own capacities in multiple resource types, and a set of tasks, each of which is associated with a value, a workload and resource requirement, the Online Scheduling for Gain Maximization (OSGM) problem is to compute an assignment of the given tasks to the servers such that the system will receive the maximum gain for the completed tasks following the assignment.

Given a set of input tasks $\mathcal{N} = \{1, 2 \ldots, N\}$, we represent each task by a tuple $i : (t_i, d_i, w_i, r_i, v_i, c_i)$, where t_i, d_i, w_i, r_i and v_i are respectively task i's arrival time, deadline for completion, workload, required resources and value (reward) which is inversely proportional to completion time if

completed by the deadline (0 otherwise), and $c_{i,j} = c_j^s + c_{i,j}^e$ is i's operation cost on server j that contains startup cost c_j^s if i is the first task starting up server j and execution cost c_i^e proportional to i's execution length l_i^j on server j, $1 \leq i \leq N$. Assume that tasks arrive in z batches at times $\{b_1, b_2, \ldots, b_z\} \subseteq \{t_1, t_2, \ldots, t_N\}$, and $b_1 < b_2 < \ldots < b_z$.

As a data center usually contains a large number of servers, for a batch of tasks arriving at any time, we follow the same assumption as [5] that the data center can supply enough servers to execute them and hence no preemption is needed. For a data-center's heterogeneous servers $\mathcal{M} = \{1, 2 \ldots, M\}$, we assume that server j at time t has capability of available resources vector a_j^t, $1 \leq j \leq M$, and on completion of task i the system will receive a (positive) gain $g_{i,j} = v_i - c_{i,j}$,where v_i is proportional to task i's completion time t_i^j on server j, $t_i^j = t_i + l_i^j$. Our OSGM problem is to compute a schedule $\pi_{i,j}$: $\mathcal{N} \rightarrow \mathcal{M}$ following the arriving batches of the tasks such that the total gain the system receives is maximized under the schedule.

Mathematically, the OSGM problem can be formulated as the following integer linear program (ILP) of computing a task-server allocation matrix $\pi_{i,j} = (x_{i,j})$, where $x_{i,j} = 1$ if $i \rightarrow j$ (task i is allocated to server j) and 0 otherwise, for the maximization of total gain. Here, constraint (3) states that task batch arrival follows an increasing time order; (4) gives the completion time of each task; (5) ensures each task to be assigned to one server; (6) ensures that the total amount of resource for running tasks on each server can not exceed its capacity, where $\sum$ and $\leq$ are element-wise on vectors r_i and a_j; (7) gives the value of task i's completion on server j; (8) gives the operation cost of task i on server j including sever launching cost and execution cost proportional to task's execution length.

$$\max \sum_{b=b_1,b_2,\ldots,b_z} \sum_{\forall i: t_i = b} \sum_{j=1}^{M} x_{i,j} \left(v_i(t_i^j) - c_{i,j} \right) \tag{1}$$

$s.t.$

$$x_{i,j} \in \{0,1\},\ \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{2}$$

$$b_1 < b_2 < \ldots < b_z \tag{3}$$

$$t_i^j = t_i^s + l_i^j,\ \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{4}$$

$$\sum_{j=1}^{M} x_{i,j} = 1,\ \forall i \in \mathcal{N} \tag{5}$$

$$\sum_{i=1}^{N} x_{i,j} r_i \leq a_j,\ \forall j \in \mathcal{M} \tag{6}$$

$$v_i(t_i^j) = v_i \text{ if } t \leq d_i \text{ and 0 otherwise},\ \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{7}$$

$$c_{i,j} = c_{i,j}^s + c_i^e(l_i^j),\ \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{8}$$

As a special case when the number of batches $z = 1$, the scheduling problem becomes offline with the objective (1) simplified to $\max \sum_{j=1}^{M} \sum_{i=1}^{N} x_{i,j} \left(v_i(t_i^j) - c_{i,j}(l_i^j) \right)$ and constraint (3) removed.

Optimizing the OSGM problem means we decide which server the specific task i is bound to and running on without knowing future tasks. Typical effective solutions to this problem are relaxation and rounding technique or genetic algorithm, which may take a long time to converge and can not make responsive decisions. Actually, for efficient and effective use in practical life, our scheduler should make scheduling decisions within reasonable latency, which is the crux.

3.3 Problem Hardness

Intuitively, the optimal gain can be obtained by enumerating all possible mapping $\pi_{i,j} : \mathcal{N} \rightarrow \mathcal{M}$, which clearly requires exponential time. Formally we establish our OSGM problem's NP-hardness by the following simple reduction from bin packing problem.

Theorem 1. *The OSGM problem is NP hard.*

Proof: The NP hardness of our OSGM problem can be easily established from the fact that even in the simple case of fixed launching cost of each server, nil value received for each task's completion, single type of resource and infinite batch size (i.e., offline setting), the OSGM problem can be reduced from the standard offline bin packing problem that is NP hard [23]. □

4 Framework of Two-Stage Scheduling

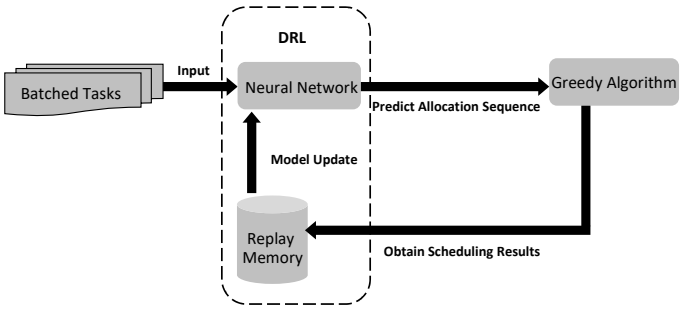


Fig. 2: Framework of DRL enhanced greedy optimization

Because the OSGM problem is NP hard, as shown in Section 3, we present in this section an approximation solution of DRL enhanced greedy approach composed of two stages of task sequencing generated by DRL followed by one-by-one task to server online placement. Task sequencing gives the order of one-by-one task-to-server placement that will generate a likely "optimal" mapping of the batch of tasks to servers based on the trained history knowledge for the current situation of server usage and arriving batch of tasks. Task placement is a greedy algorithm with a performance guarantee that is strictly online without knowing the information of other tasks in the current batch and therefore its performance may be improved if the placement follows the order recommended by DRL based on the knowledge of batch information. Our DRL-greedy allied approach combines the merits of DRL for best placement sequence prediction for a batch of tasks and greedy algorithm for assurance of performance (competitive ratio) guarantee in the one-by-one placement.

We outline the framework of our DRL-greedy online scheduling scheme in Fig. 2. It consists of two stages: one is DRL composed of neural networks and a replay memory

deciding $\pi_i : \mathcal{N}_b \to \mathcal{N}_b$, which denotes the allocation order of task i in batch $\mathcal{N}_b$, the other is an online greedy based algorithm deciding $\pi_{i,j} : \mathcal{N}_b \to \mathcal{M}$, denoting the allocation of task i to server j.

When a batch of tasks arrives, the scheduler decides the allocation sequence first and then places them one by one to the servers in a greedy manner following the sequence. It works as follows:

In the first stage, the neural network in DRL takes the system state including capabilities of all servers and the information of newly arriving tasks as input, and decides task sequence π_i as output based on the current trained policy. The policy of DRL is updated by selecting a mini-batch of historical experience from replay memory and then automatically learning to generate a task sequence that leads to better performance.

In the second stage, the greedy algorithm makes the decision of $\pi_{i,j}$ applying a greedy approach of maximizing the gain for the placement of task i following the order in the sequence generated by DRL. After placing tasks as $\pi_{i,j}$, we obtain a reward for training DRL and record the experience tuples in replay memory, including current system state, action of task sequencing, the aforementioned reward and next time system state.

When DRL has been successfully trained, we can deploy our DRL–greedy optimization algorithm in a practical cloud platform for real time scheduling, avoiding the long time convergence as the typical genetic algorithm or relaxation and rounding technique does. Because our greedy algorithm ensures a competitive ratio with any task sequence π_i, the worst case of gain achieved by DRL-greedy online scheduling also has a performance guarantee. This is the reason why we do not directly employ DRL to decide task placement $\pi_{i,j}$, which suffers from unstable performance especially when the distribution of training data is not consistent with the one in the production environment. Besides, DRL can utilize the batch information, recommend the greedy algorithm placing tasks one by one in a better sequence to enhance performance and fill the gap between $\frac{1}{1+\kappa}$ and the optimal competitive factor 1.

5 Algorithm Description

The main algorithm of our DRL-Greedy online scheduling for task batches is given in Alg. 1. The greedy algorithm makes an one-by-one task-to-server placement decision $\pi_{i,j}$ follows the order predicted by DRL, and the corresponding effect will be stored for training neural networks of DRL.

The procedures of Task Sequencing and Task Allocation are described below.

5.1 Task Sequencing

We apply DRL for task sequencing built on the actor-critic framework which has been shown capable of achieving long-term performance optimization [24]. As shown in Alg. 1, when a new batch of tasks arrives at time b_k, DRL observes the system state S_{b_k}. Based on S_{b_k}, the actor network generates the task scheduling sequence A_{b_k} and the performance of A_{b_k} will be predicted by critic network. After it, our greedy algorithm makes scheduling decisions and the reward R_{b_k} is produced. We record the historical trajectories $(S_{b_k}, A_{b_k}, R_{b_k}.S_{b_{k+1}})$ in replay memory, which will be used for model training. These variables are defined as follows:

Algorithm 1 Procedure of DRL-Greedy Online Scheduling

Input: A batch of tasks arriving at time b_k
Output: Task placement policy $\pi_{i,j}$
▷ Task Sequencing :
1: Decide task sequence π_i by the actor network in DRL as ① in Fig. 3.
▷ Task Allocation :
2: Allocate tasks to servers as $\pi_{i,j}$ one by on in the given order generated by DRL by the greedy algorithm as ② in Fig. 3.
▷ DRL Training :
3: Capture current system state by information collector as ③ in Fig. 3 after allocation decisions have been made.
4: Write historical experience $(S_{b_k}, A_{b_k}, R_{b_k}.S_{b_{k+1}})$ into replay memory as ④ in Fig. 3.
5: Select a mini-batch from replay memory to update first the parameters of critic network, which makes long term performance prediction of task sequencing, and then the parameters of actor network as ⑤ in Fig. 3.

1) **State Space**: In our context, the system state when batched tasks arrive at time b_k can be defined as Eq. 9, where S_{b_k} contains the pending tasks $\mathcal{I}_{b_k} = \{i' | i' \text{ is pending}\}$ and capabilities $\mathcal{A}_{b_k} = \{a_j^{b_k} | 1 \leq j \leq M\}$ stating the available resource of all servers at time b_k.

$$S_{b_k} = (\mathcal{I}_{b_k}, \mathcal{A}_{b_k}). \tag{9}$$

2) **Action Space**: An action A_{b_k} for deciding the task allocation sequence are made based on the observed system state S_{b_k}, which is stated in Eq. 10.

$$A_{b_k} = \{\pi_i | \forall i : t_i = b_k\}. \tag{10}$$

3) **Transition Probability**: In online settings, the subsequent state $S_{b_{k+1}}$ is determined by S_{b_k}, A_{b_k} and the task batch arriving at future time b_{k+1}.

4) **Reward**: After we employ greedy algorithm for task allocation, π_i and $\pi_{i,j}$ are all determined and the system receives the gain of completing tasks $\mathcal{I}_{b_k}$ defined as $g(\mathcal{I}_{b_k}) = \sum_{i \in \mathcal{I}_{b_k}} \left(\sum_{j=1}^{M} x_{i,j} \left(v_i(t_i^j) - c_{i,j} \right) \right)$. For DRL, we define reward function R_{b_k} as Eq. 11, which is the cumulative gain up to the current time b_k divided by that before b_k.

$$R_{b_k} = \frac{\sum_{k'=1}^{k} g(\mathcal{I}_{b'_k})}{\sum_{k'=1}^{k-1} g(\mathcal{I}_{b'_k})}. \tag{11}$$

As illustrated in Fig. 3, the actor-critic DRL framework consists of an actor network for control and a critic network for evaluation. To reduce the training oscillation, we adopt an additional neural network for both the actor network and critic network, which have the same structure as the original ones and provide relatively stable values. For the sake of clarity, we refer to the original actor and critic network as online policy and Q network respectively. The additional actor and critic network are target policy and Q network

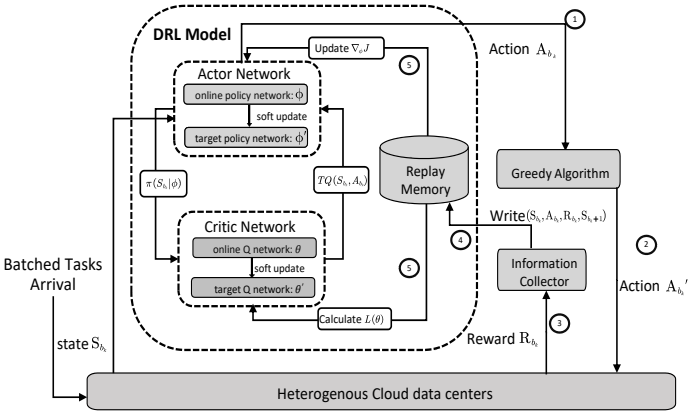


Fig. 3: The DRL framework

respectively. The weights of these target networks are then updated by having them slowly track the learned online networks as the following Eq. 12, where $\theta, \phi, \theta', \phi'$ is the parameters of online Q, online policy, target Q and target policy network respectively and $\tau \ll 1$. Since τ is relatively small, the target values are constrained to change slowly every time, greatly improving the learning stability.

Algorithm 2 DRL Training

Input: Super-parameters including replay memory capacity, batch size, actor network learning rate, critic network learning rate and soft update parameter τ
Output: Optimized parameters θ, ϕ

```
for each episode do
    for Each task batch arriving in time b_k ∈ {b_1, b_2, ..., b_z} do
        Obtain action A_{b_k} via online policy network for the given system state S_{b_k}.
        Apply Alg. 3 to generate a scheduling policy based on A_{b_k}.
        Perform this policy and store the experience tuple (S_{b_k}, A_{b_k}, R_{b_k}, S_{b_{k+1}}) in replay memory.
        Select a random mini-batch of experience tuples from replay memory.
        Calculate Q-value as Eq. 13.
        Update parameters of the online policy network and online Q network with Eq. 15 and Eq. 16 respectively.
        Soft-update parameters of the target networks using Eq. 12.
    end for
end for
```

$$\begin{aligned} \theta' &\leftarrow \tau\theta + (1-\tau)\theta', \\ \phi' &\leftarrow \tau\phi + (1-\tau)\phi'. \end{aligned} \tag{12}$$

Note that R_{b_k} is the immediate reward of state-action pair (S_{b_k}, A_{b_k}) and $\rho\left(S_{b_{k+1}} \mid \phi\right)$ denotes the predicted action output by the actor network given input $S_{b_{k+1}}$. In addition, $Q\left(S_{b_{k+k'}}, \cdot \mid \theta\right)$ denotes the long-term reward at time $b_{k+k'}$ of the state-action pair $\left(S_{b_{k+k'}}, A_{b_{k+k'}}\right)$, which is estimated by the critic network.

The parameters of the actor network and critic network are updated through periodic training. At each training iteration, we randomly select samples from the replay memory to update the network parameters. For a state-action pair (S_{b_k}, A_{b_k}), the critic network attempts to predict the long-term reward after multiple steps s guided by current policy Ψ. Given the discount factor of time $\gamma \in (0,1)$, the long term reward is measured by Q-value in Eq. 13, which is the expectation of cumulated rewards.

$$Q^{\Psi}\left(S_{b_k}, A_{b_k}\right) = \mathbb{E}\left[\sum_{s=0}^{\infty} \gamma^s R_{b_{k+s}}\right]. \tag{13}$$

In the replay memory, we combine the experience tuples in consecutive time slot and obtain complete trajectories $(S_1, A_1, R_1, S_2, A_2, R_2, \ldots)$ following policy π. Given the impracticality of obtaining the complete trajectories in real-world cloud environments, we exploit the information in m steps as a compromise to better estimate the long term reward. Then, by the target Q network, we compute the cumulated reward of the sampled historical trajectories as Target Q-value in Eq. 14.

$$\begin{aligned} TQ\left(S_{b_k}, A_{b_k}\right) &= R_{b_k} + \gamma \cdot R_{b_{k+1}} + \gamma^2 \cdot R_{b_{k+2}} + \cdots + \\ &\gamma^{k'-1} \cdot R_{b_{k+k'-1}} + \gamma^m \cdot \mathbb{E}\left[Q^{\Psi}\left(S_{b_{k+k'}}, \pi\left(S_{b_{k+k'}} \mid \phi\right) \mid \theta'\right)\right]. \end{aligned} \tag{14}$$

Using the mean squared error as the loss function for critic network, we have the following difference between Target Q-value in Eq. 14 and Q-value in Eq. 13, which will be used for updating parameters of the online Q network:

$$L(\theta) = \mathbb{E}\left[\left(Q^{\Psi}\left(S_{b_k}, A_{b_k} \mid \theta\right) - TQ\left(S_{b_k}, A_{b_k} \mid \theta\right)\right)^2\right]. \tag{15}$$

For training the actor network, its target policy network will produce the estimated optimal action $A_{b_k} = \Psi\left(S_{b_k} \mid \phi\right)$, and for given A_{b_k}, the parameters of its online policy network are updated with the policy gradient from the back propagation of value function (i.e., critic network). With the policy gradients of the expected long-term reward $\nabla_\phi Q^{\Psi}(S_{b_k}, A_{b_k} \mid \theta)$ being the product of the gradients of state action pair Q-value with respect to action and the gradient of action with respect to Ψ[24], we update the parameters of the actor network with policy gradient $\nabla_\phi J$ according to the following equation:

$$\begin{aligned} \nabla_\phi J &\approx \mathbb{E}\left[\nabla_\phi Q^{\Psi}(S_{b_k}, A_{b_k} \mid \theta)\right] \\ &= \mathbb{E}\left[\nabla_{A_{b_k}} Q^{\Psi}(S_{b_k}, A_{b_k} \mid \theta) \cdot \nabla_\Psi \rho(S_{b_k} \mid \phi)\right]. \end{aligned} \tag{16}$$

We explain our DRL training in detail in Alg. 2.

5.2 Task Allocation

We now give our greedy algorithm to schedule tasks in the Task Allocation part of Alg. 1. After DRL generates the allocation order, our greedy algorithm will iterate over the pending tasks following the given sequence. When dealing with task i, the algorithm greedily chooses the server, which will produce the maximum marginal gain if we allocate task i to this server. As we have seen in Alg. 3, a task will be irrevocably allocated to a server during each iteration.

That is, after DRL model performs action A_{b_k}, Alg. 3 makes action A'_{b_k} defined in Eq. 17.

$$A'_{b_k} = \{\pi_{i,j} | \forall i : t_i = b_k, 1 \le j \le M\}. \tag{17}$$

The $\frac{1}{1+\kappa}$ competitive ratio will be proven in the next section Algorithm Analysis.

Algorithm 3 Greedy Based Task Allocation Algorithm

Input: The task batch arrives at b_k
Output: Decisions for allocating tasks to run on a which server j

1: **for** each pending task $i \in \mathcal{I}_{b_k}$ **do**
2: **for** each server j **do**
3: Calculate the incremental gain $g_{i,j} = v_i(t_i^c) - c_{i,j}^s - c_i^e(l_i^j)$, if we assign task i to server j, where $c_{i,j}^s = 0$ if task i is not the first task running on j.
4: **end for**
5: Obtain $j' = argmax\, g_{i,j}$.
6: Makes a decision that $x_{i,j'} = 1$.
7: **end for**

6 Algorithm Analysis

In this section, we will prove the competitive ratio of Alg 3, i.e., the gain of completing tasks using our online algorithm compared to the optimal offline one, and analyze the time complexity.

6.1 Competitive Ratio analysis

6.1.1 Basic idea

As the competitive ratio of our DRL-Greedy online scheduling algorithm is guaranteed by that of the greedy online task placement algorithm. We only need to show the competitive ration of our greedy algorithm.

Let ALG and OPT be our Alg. 3 and the optimal algorithm, $ALG(P_N)$ and $OPT(P_N)$ be the total gain of our greedy algorithm and the optimal algorithm respectively. We apply structural induction [25] to establish $OPT(P_N) \le (1+\kappa)ALG(P_N)$, which is similar to mathematical induction, but works in the problem domain of recursively defined structures. Denoting by P_{N-K} the OSGM problem of scheduling the remaining task set $\{i' | K+1 \le i' \le N\}$ after the first K task are scheduled, we can solve P_{N-K} by allocating first task $K+1$ and then all tasks arriving after it using a recursive call on P_{N-K-1}. Because of the backward recursion, the base case is concluded to allocate the last task, i.e. task N when all tasks before it $\{i' | 1 \le i' < N\}$ have been allocated. So the proof goes by first showing the competitive ratio for the base case P_1 (task N), and then making recursive induction to show it holds for P_{N-K} for $K = N-2, N-3, \ldots, 0$.

6.1.2 Proof

We first establish the following lemma and then prove the competitive ratio of our algorithm based on the lemma using induction.

Lemma 2. *$OPT(P_{N-K}) \le OPT(P_{N-K-1}) + (1+\kappa)g_{K+1,j_{K+1}}$, where $\kappa = \max_{i\in\mathcal{N}, j\in\mathcal{M}} \frac{c_{i,j}^s}{g_{i,j}}$.*

Proof: Denote by $\mathcal{N}_j = \{i | \forall i : x_{i,j} = 1\}$ the set of tasks running on server j, by $g(\mathcal{N}_j)$, $v(\mathcal{N}_j)$ and $c(\mathcal{N}_j)$ the total gain, value and cost for executing all tasks in $\mathcal{N}_j$ on server j, and by $i \to j$ allocating task i to server j. For problem P_{N-K}, without loss of generality we assume that the first task $K+1$ is allocated to server j_{K+1} by OPT and OPT produces the following solution:

$$\{\mathcal{N}_1, \cdots, \mathcal{N}_{j_{K+1}}, \ldots, \mathcal{N}_M\}. \tag{18}$$

Let $\mathcal{N}^-_{j_{K+1}} = \mathcal{N}_{j_{K+1}} / \{K+1\}$. Because allocating a task on any server will result in positive gain and $OPT(P_{N-K})$ differs from $OPT(P_{N-K-1})$ only in the addition of the allocation of task $K+1$ to server j_{k+1}, we have

$$OPT(P_{N-K}) - OPT(P_{N-K-1}) = g(\mathcal{N}_{j_{K+1}}) - g(\mathcal{N}^-_{j_{K+1}}).$$

□

Because

$$v(\mathcal{N}_{j_{K+1}}) - v(\mathcal{N}^-_{j_{K+1}}) = v_{K+1}$$

and

$$c(\mathcal{N}_{j_{K+1}}) - c(\mathcal{N}^-_{j_{K+1}}) \le c^e_{K+1,j_{K+1}} + c^s_j,$$

we have

$$\begin{aligned} & g(\mathcal{N}_{j_{K+1}}) - g(\mathcal{N}^-_{j_{K+1}}) \\ &= v(\mathcal{N}_{j_{K+1}}) - v(\mathcal{N}^-_{j_{K+1}}) - (c(\mathcal{N}_{j_{K+1}}) - c(\mathcal{N}^-_{j_{K+1}})) \\ &\le v_{K+1} - c^e_{K+1,j_{K+1}} - c^s_{j_K} = g_{K+1,j_{K+1}} \\ &= \frac{v_{K+1} - c_{K+1,j_{K+1}} + c^s_{j_{K'}}}{g_{K+1,j_{K+1}}} \times g_{K+1,j_{K+1}} \\ &\le (1 + \frac{c^s_{j_{K+1}}}{g_{K+1,j_{K+1}}}) g_{K+1,j_{K+1}}. \end{aligned} \tag{19}$$

Letting $\kappa = \max_{i\in\mathcal{N}, j\in\mathcal{M}} \frac{c^s_{i,j}}{g_{i,j}}$, we immediately get $OPT(P_{N-K}) \le OPT(P_{N-K-1}) + (1+\kappa)g_{K+1,j_{K+1}}$. Hence the lemma holds.

Now, we prove the competitive ratio based on Lemma 2 using induction.

Theorem 3. *Alg. 3 has a competitive ratio of $\frac{1}{1+\kappa}$ to the optimal solution, where $\kappa = \max_{i\in\mathcal{N}, j\in\mathcal{M}} \frac{c^s_{i,j}}{g_{i,j}}$.*

Proof: Let $ALG(P_{N-K})$ and $OPT(P_{N-K})$ be the solutions of Alg. 3 and the optimal algorithm for allocating $N-K$ tasks. We make induction on K to show $OPT(P_{N-K}) \le (1+\kappa)ALG(P_{N-K})$ for any K.

For $K = N-1$ (basis), i.e., for allocating the single task $K+1 = N$ when all tasks $K, K-1, \ldots, 1$ have been allocated, the theorem clearly holds because $OPT(P_1) = ALG(P_1)$.

Assume that it holds for any $K+1 \le N-2$, i.e., for allocating $N-K-1$ tasks $N, N-1, \ldots, K+2$, we have

$$OPT(P_{N-K-1}) \le (1+\kappa)ALG(P_{N-K-1}). \tag{20}$$

Now considering the case of value K,we need to allocate $N-K$ tasks $N, N-1, \ldots, K+2, K+1$, which is realized incrementally by allocating the additional task $K+1$ on the basis of the allocation of $N-K-1$ tasks in the previous hypothesis step. Assume that Alg. 3 allocates task $K+1$ on

server j'_{K+1}. Because the algorithm chooses server j'_{K+1} by the greedy strategy that brings the largest gain, we have

$$g_{K+1,j'_{K+1}} \geq g_{K+1,j}, \forall j \neq j'_{K+1}. \tag{21}$$

Applying Eq. 20, 21 and Lemma 2 yields

$$\begin{aligned} &OPT(P_{N-K}) \\ &\leq OPT\left(P_{N-K-1}\right) + (1+\kappa) g_{K+1,j_{K+1}} \\ &\leq (1+\kappa) ALG\left(P_{N-K-1}\right) + (1+\kappa) g_{K+1,j'_{K+1}} \\ &= (1+\kappa) ALG(P_{N-K}). \end{aligned} \tag{22}$$

This completes the proof. □

6.1.3 Comparison with state-of-the-art algorithms

Researchers in [5] present a similar work to maximize the total gain of tasks without considering execution cost. It assumes that $\delta \sum c^s_{K',j_{K'}} \geq \sum_{i\in\mathcal{N}} v_i^{max} \geq \sum_{i\in\mathcal{N}} v_i^{min} \geq \rho \sum c^s_{K',j_{K'}}$, where v_i^{min} is the minimum value achieved by completion of task i before deadline, and v_i^{max} is the maximum one. It achieves an expected competitive ratio $\frac{\rho-2}{\delta}$. We now show that our competitive ratio in expectation is lower bounded by $\frac{\rho}{\rho+1}$ which is at least the same as $\frac{\rho-2}{\delta}$.

Theorem 4. *The expected competitive ratio of Alg. 3 is lower bounded by* $\frac{\rho}{\rho+1}$.

Proof: We derive that $\mathbb{E}(1 + \max_{\mathrm{i}\in\mathcal{N},\mathrm{j}\in\mathcal{M}} \frac{c^s_{i,j}}{g_{i,j}}) = 1 + E(\max_{\mathrm{i}\in\mathcal{N},\mathrm{j}\in\mathcal{M}} \frac{c^s_{i,j}}{g_{i,j}}) \leq 1 + \frac{\rho \sum c^s_{K',j_{K'}}}{\sum_{i\in\mathcal{N}} v_i^{min}} \leq 1 + \frac{1}{\rho}$, and thus the expected competitive ratio is lower bounded by $\frac{\rho}{\rho+1}$. Note that our algorithm can also handle the case that each tasks has different parallelism degrees at extra computation cost of enumerating all possible task partitions like what algorithm in [5] does. □

Theorem 5. *The expected competitive ratio of Alg. 3 is better than that of the algorithm proposed in [5].*

Proof: Because $\delta \geq \rho \geq 2$, it is easily verified that $\delta \geq \rho - 1 - \frac{2}{\rho}$, which means $\frac{\rho}{\rho+1} \geq \frac{\rho-2}{\delta}$. Therefore, theorem 5 can be proved. □

6.2 Time Complexity Analysis

Furthermore, we analyze the time complexity of Alg. 1 as below. Denoting by $\mathcal{L}$ the set of neural network layers, and by L_i the number of parameters of layer i, we give the training time complexity in the following theorem 6.

Theorem 6. *. The complexity of training Alg. 1 is $\mathcal{O}(n \times It \times \Lambda \times M)$, where n is the number of training samples, It is the number of iterations and $\Lambda = \sum_{0<i<|\mathcal{L}|} (L_i \times L_{i+1})$.*

Proof: It is not hard to verify that the number of parameters can be calculated as $\sum_{0<i<|\mathcal{L}|} (\mathrm{L}_i \times \mathrm{L}_{i+1})$. For a single training sample in an iteration, the complexity of Alg. 3 is $\mathcal{O}(M)$. Therefore, if Λ denotes the number of neural network's parameters, the theorem holds. □

Furthermore, the inference time complexity of our DRL-greedy online scheduling is calculated as $\mathcal{O}(\Lambda \times \mathrm{M})$ since it does not involve the steps of gradient descent.

7 Experimental Evaluation

In this section, we present our experimental results in both simulation and real application environments. We first provide our basic experiment settings, and describe the baselines for evaluation and research questions. Then we show implementation results and performance comparisons with respect to these research questions.

7.1 Experiment Setup

7.1.1 Settings of Simulation Environment

We simulate a cloud environment with 100 heterogeneous servers. Thirty of them have a 32-core CPU of 4 GHz (clock speed), thirty of them have a 16-core CPU of 3.2 GHz, and others have a 16-core CPU of 2.2 GHz. All the servers have the same memory capacity of 32 GB. In addition, we assume the launching cost of each server is proportional to the resource capacity. A group of previous works conduct experiments on the public dataset, Google cluster workload traces 2019[26], [27], which contains the submission time and sensitivity to the latency of each task. Different latency sensitivity indicates that the value of completion declines at different rates as the completion time is gradually longer. There are three different levels of completion time sensitivity. When tasks are completed before deadline, we define three functions of value as : $v_i = v_i^{max} - 0.5t$, $v_i = v_i^{max} - t$ and $v_i = v_i^{max} - 2t$, where v_i^{max} is the maximum value achievable and t is the completion time. We assume each task's requested CPU cores and memory size are uniformly distributed from 1 to 8 and in the range of $[0.5, 2]$ respectively. The task execution cost of a unit time is proportional to the CPU clock speed of servers and the size of required resource. Server launching cost is proportional to the resource capacity. As for the setting of DRL, both the actor network and critic network have two hidden layers, with 50 and 30 neurons, respectively. ReLU is the activation function and that as the output function of each actor network. The batch size for DRL training is 64 and maximum capacities of replay memory is 10,000. The initial learning rates of the critic network and actor network are $1e-3$ and $1e-4$, respectively. The discount rate for future rewards γ is 0.9. For the soft-update of target networks, we set the τ as $2e-3$. Lastly, the Adam optimizer is used to optimize the neural networks. The DRL agent is trained for 6500 episodes in our experiment.

7.1.2 Settings of Real Application Environment

To further validate the practicality and applicability of our algorithm, we build a real application environment for applying our scheduling algorithm. The application contains a stream of 3,000 face detection tasks requested from different police stations to find the missing persons and criminals. The tasks are submitted randomly, forming batches of size following Poisson distribution with average batch size 10, and required to be completed online as each batch arrives. We run the application under the scheduling by our algorithm (and the baseline schedulers) on heterogeneous cloud servers (virtual machines) of different scales ranging from 20 to 100 servers. These servers are composed of 5 different types in the ratios of 20%, 30%, 20%, 20% and 10% respectively to reflect the genuine heterogeneity: (1) 2 i7-6700

vCPU and 4 GiB memory; (2) 2 Intel Xeon 8369B vCPU and 4 GiB memory; (3) 8 Intel Xeon 8163 vCPU and 8 GiB memory; (4) 8 Intel Xeon 8269CY vCPU and 16 GiB memory; (5) 16 Intel Xeon 8269CY vCPU, 32 GiB memory and 1 NVIDIA A10 GPU. The required computing resource (CPU and GPU) of each task is randomly generated based on the uniform distribution as in $[1, 4]$, and the required memory size is proportional to the video length of the task. The maximum value of task completion time satisfies normal distribution $\mathcal{N}(10, 20)$. Similar to the simulation settings, there are three completion time sensitivity levels and each task is assigned with a random level. We also use the methods described in Section 7.1.1 to measure task completion value and cost for each task. We run our scheduler and each baseline on a single type-(5) server and the application (batches of tasks) on the set of servers allocated by the scheduler.

7.2 Baselines

- **First-Fit**[28]. Each time when a new task arrives, it tries to put it into the earliest opened server that can accommodate it. If none of the opened servers has enough resource capacity to accommodate the new task, then a new server will start up.
- **OptRA**[21]. According to the shortest job first policy, this algorithm allocates the tasks to the servers with the highest efficiency as long as the constraints are not violated. In this way, the task completion time will be reduced. However, it does not consider the server launching cost, we modify it as: when the running servers do not have enough resource to execute tasks, a new server with the highest CPU clock speed will start up.
- **OSSA** [5]. This algorithm is proposed to optimize the bi-objective of gain and user satisfaction rate. We only focus on the gain in this experiment.
- **HMAO** [7]. The HMAO algorithm employs genetic algorithm to optimize the server utilization. We modify its objective to cater for our needs. To be more specific, in this evaluation, the objective of genetic algorithm in HAMO is changed to the combination of value and the corresponding operation cost, where the communication cost is not included. We set the population size, crossover probability and mutation probability to be 16, 0.25 and 0.25 respectively, which are the same as the parameters in [7].
- **DRL** [8], [9]. Each time when a batch of task arrives, the typical DRL based algorithm directly produces the task-to-server allocation plan. We also adopt the actor-critic framework and all the super-parameters is the same as our DRL-greedy online scheduling algorithm.

7.3 Evaluation Metrics

The research questions guiding the remainder of the paper are: (**RQ1**) Can the proposed algorithm achieve high performance in comparison with the baselines? (**RQ2**) Is our algorithm more robust than the traditional DRL algorithm in changing cloud environments? (**RQ3**) Can our algorithms sustain high scalability while solving a large scale problem? (**RQ4**) Although the theoretical lower bound is given, how far are the actual results of our algorithm from the optimal one? (**RQ5**) What is the effect of our algorithm in real application environment?

7.4 Implementation in Simulation Environment

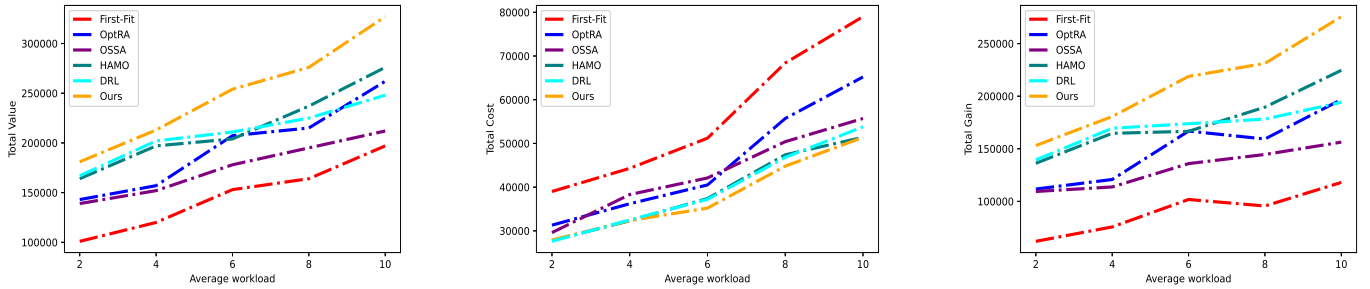
We answer **RQ1** ~ **RQ4** with our implementation results in a simulation environment with google cluster data traces.

7.4.1 High Performance

We first answer **RQ1**. In our simulations, we assume that the v_i^{max} follows normal distribution $\mathcal{N}(\eta, 2\eta)$ and the corresponding workload is proportional to it, which means the average task workload is approximately η. We now study the performance of our DRL-greedy online scheduling with different average task workloads, shown in Fig. 4a to Fig. 4c. In Fig. 4a, we compare the achieved task completion value of our scheduling algorithm with First-Fit, OSSA, OptRA and HMAO. Our scheduling algorithm outperforms the First-Fit algorithm by 63.1 percent, OSSA algorithm by 38.3 percent, OptRA algorithm by 24.4 percent, DRL algorithm by 17.75 percent and HAMO algorithm by 14.1 percent respectively on average. Moreover, when the required execution time is large, the improvement of our algorithm becomes significant as well. Besides, we can see that in Fig. 4b, the DRL-greedy online scheduling algorithm outperforms the First-Fit algorithm by 41.9 percent, OSSA algorithm by 12.2 percent, OptRA algorithm by 16.7. percent, DRL algorithm by 2.8 percent and HAMO algorithm by 2.1 percent respectively on average. Jointly considering the value and cost, our DRL-greedy online scheduling again produces the best results, which is demonstrated in Fig. 4c. It at least outperforms other baselines by 13.4 percent. These results reveal that our algorithm stably outperforms others under different situations.

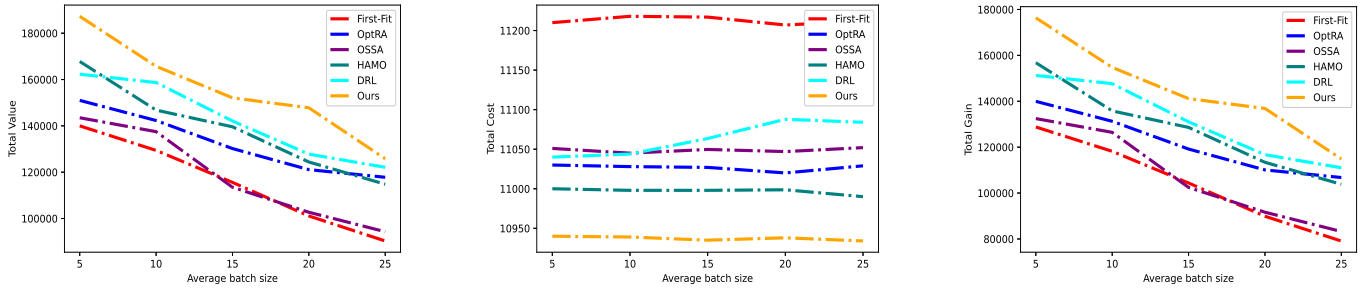
To facilitate more comprehensive comparisons with the previous work, we now consider the situation that the number of available servers is too small to simultaneously run all the tasks of the same batch. In our experiment, only 10 servers can be used, and the ratio of different types of servers is the same as we discuss in section 7.1. We assume the task arrival time satisfies Poisson distribution with λ denoting the average number of tasks that arrived a unit of time. Fig. 5a to Fig. 5c evaluates the performance of our algorithm as the average batch size(number of tasks in the same batch) increases and the total number of the tasks stays the same. As shown in Fig. 5b, the cost produced will not change significantly. Because many tasks have to wait for execution, Fig. 5a and 5c show the achieved value and gain decrease almost linearly as λ increases. In Fig. 5c, our DRL-greedy online scheduling algorithm outperforms the First-Fit algorithm by 115.7 percent, OSSA algorithm by 53.9 percent, OptRA algorithm by 35.8 percent, DRL algorithm by 10 percent and HAMO algorithm by 17.7 percent on average. We remark that when the number of tasks in the same batch is the smallest, the performance gap between baselines and our algorithm is minimal.

We notice that in online settings, the future information about task arrivals and demands is unknown, and thus we need to consider these future uncertainties carefully when making decisions.Our algorithm deploys the DRL module to predict future events, which enables the performance of our algorithm to be superior to those without using



(a) Value($) versus average workload (b) Cost($) versus average workload (c) Gain($) versus average workload

Fig. 4: Performance comparison wrt average workload for average batch size 10



(a) Value($) versus average batch size (b) Cost($) versus average batch size (c) Gain($) versus average batch size

Fig. 5: Performance comparison wrt average batch size for average workload 6

DRL. However, only deploying DRL to make decisions will make our scheduler extremely sensitive to environmental changes. Our algorithm deploys a greedy optimization strategy to provide the worst-case performance guarantee that bounds the algorithm's sensitivity and enable our algorithm to have better robustness than those relying only on DRL as shown below.

7.4.2 Robustness

For **RQ2**, we assume a situation that servers may break down and can not restart when average batch size is 20 and average workload is 6. Fig. 6 depicts the performance comparison of our DRL-greedy online scheduling algorithm and only DRL, which makes the allocation decisions directly when the probability that a server breaks down increases. In extreme cases that 20 percent of servers suddenly fail to work, our DRL-greedy scheduling outperforms the typical DRL scheduling by nearly 52 percent. It is obvious that DRL-greedy online scheduling algorithm are more robust to the environment change.

7.4.3 Scalability

Concerning **RQ3**, we study the performance of our online algorithm when solving a large scale problem. From Fig. 7a to 7c, we can see that when the total number of tasks increases, the increments of value, cost and gain by our DRL-greedy online scheduling algorithm are nearly the same, which demonstrates that our algorithm can solve problems at large scale effectively. Our algorithm on average

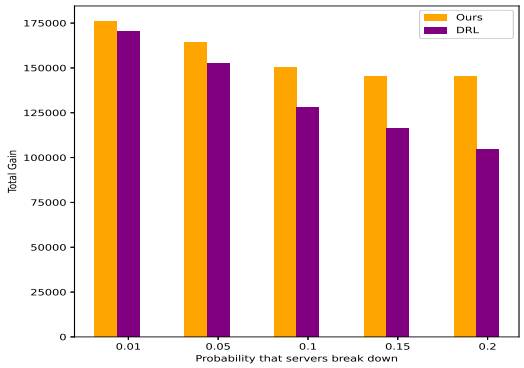
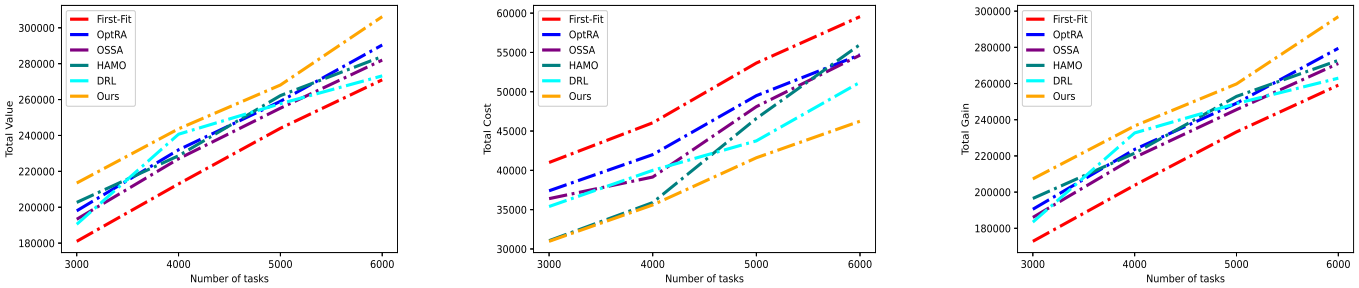


Fig. 6: Evaluation of Robustness

outperforms the First-Fit algorithm by 16 percent, OSSA algorithm by 8.68 percent, OptRA algorithm by 6.3 percent, DRL algorithm by 8.24 percent and HAMO algorithm by 6 percent respectively.

7.4.4 Gap from the optimum

Next, we answer **RQ4**, studying the performance gap between our DRL-greedy online scheduling and the optimal offline gain under different settings. Enumerating all possible solutions, we use the brute force approach to obtain the optimal offline gain. Because of the hardness of solving this NP-hard problem, we limit the number of tasks to be 40 and use 10 servers in our experiments. Fig. 8 shows that the ratios of value, cost and gain are around 0.87, 0.86 and 0.86



(a) Value($) versus number of tasks (b) Cost($) versus number of tasks (c) Gain($) versus number of tasks

Fig. 7: Performance comparison wrt number of tasks for average workload 6 and batch size 10

with varying number of tasks respectively. With the number of submitted tasks increasing, the actual competitive ratio of value shows a slight decrease. We believe this may be because with more tasks to be scheduled, a wrong allocation decision made before will produce more influence on the future allocation.

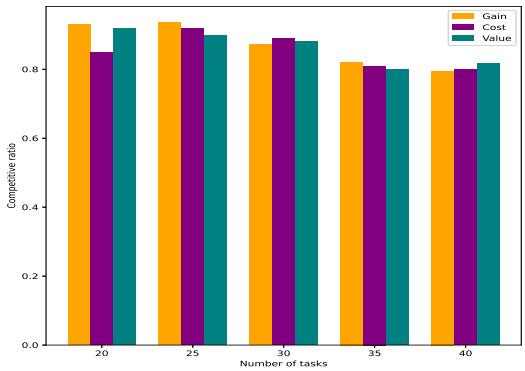


Fig. 8: Competitive ratio of DRL-greedy online scheduling algorithm wrt number of submitted tasks

7.5 Implementation in Real-World Scenarios

Finally, to demonstrate the performance of our scheduling algorithm in real-world scenarios, we answer **RQ5** with implementation results in the real application environment described in Section 7.1.2. For the given stream of arriving task batches, we apply our scheduling algorithm and all baselines to generate task-to-server allocation schemes on different numbers of available cloud servers between 20 to 100 to achieve the same goal of gain maximization. Their performances are plotted in Fig. 9. It can be seen that on average our algorithm outperforms First-Fit algorithm by 138.57 percent, OptRA algorithm by 26.17 percent, OSSA algorithm by 24.71 percent, HAMO algorithm by 11.31 percent and DRL algorithm by 29.13 percent respectively.

We also measure the scheduling latency (computation overhead) of our scheduler and all baselines by recording the time that each scheduler takes to generate a task-to-server allocation scheme when a batch of tasks arrives. Fig. 10 shows the cumulative distributions of scheduling latencies in the whole online experiment process. Because the average scheduling latency for our algorithm is about 25ms

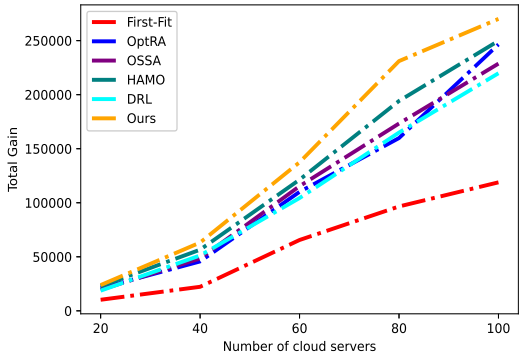


Fig. 9: Total gain achieved for running real-world applications under different scheduling schemes

and the interval between scheduling events (gap between two batches of tasks) is typically in the scale of seconds [29], our algorithm dose not impose a measurable delay on scheduling. However, the genetic algorithm, like HAMO, fails to be responsive in online scheduling environments.

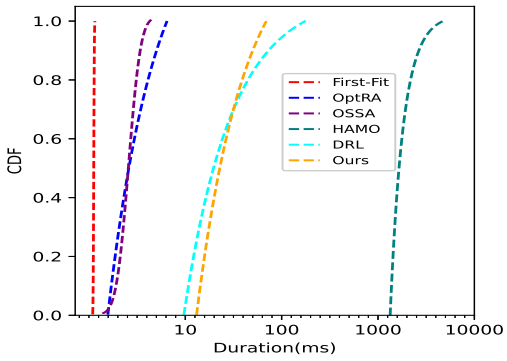


Fig. 10: Scheduling latency

8 Conclusion

This paper studies the problem of online task scheduling in HPC systems for the objective of maximizing the total system gain, defined as the value of completed tasks minus system's operation costs. To address this problem, we proposed a DRL (deep reinforcement learning) enhanced

greedy optimization algorithm for online scheduling of task batches on heterogeneous servers, which combines the merits of both greedy optimization providing worst-case performance guarantee and DRL enabling learning via interaction with the true environment. We analyzed the competitive ratio, which improves the existing result for the same problem[5], and time complexity of the algorithm. We further conducted extensive experiments in both simulation environment based on the dataset of Google cluster trace and real application environment of 3,000 face detection tasks submitted randomly in batches using 20~100 cloud servers. The experiment results validated our theoretical analysis and demonstrate that our solution outperforms the existing state-of-the-art results. Our proposed approach of combining machine-learning and greedy optimization techniques provides a promising way for solving large-scale hard optimization problems more effectively than applying these techniques alone. In the future, we will further study how to schedule workflow jobs in more sophiscated settings like [30].

ACKNOWLEDGEMENT

This work is supported by Key-Area Research and Development Plan of Guangdong Province #2020B010164003. The corresponding author is Hong Shen.

REFERENCES

[1] S. Li, M. Song, P.-J. Wan, and S. Ren, "A 2-approximation algorithm for scheduling parallel and time-sensitive applications to maximize total accrued utility value," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 1864–1878, 2016.

[2] J. Gentry, C. Denninnart, and M. A. Salehi, "Robust dynamic resource allocation via probabilistic task pruning in heterogeneous computing systems," in *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*. IEEE, 2019, pp. 375–384.

[3] R. Ren, Y. Zhu, C. Li, and X. Tang, "Interval job scheduling with machine launch cost," *IEEE Trans. Parallel Distributed Syst.*, vol. 31, no. 12, pp. 2776–2788, 2020.

[4] W. V. Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester, "Trends in worldwide ICT electricity consumption from 2007 to 2012," *Comput. Commun.*, vol. 50, pp. 64–76, 2014.

[5] B. Zheng, L. Pan, and S. Liu, "Market-oriented online bi-objective service scheduling for pleasingly parallel jobs with variable resources in cloud environments," *Journal of Systems and Software*, vol. 176, p. 110934, 2021.

[6] J. Yu and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in *2006 Workshop on Workflows in Support of Large-Scale Science*, 2006, pp. 1–10.

[7] X. Gao, R. Liu, and A. Kaushik, "Hierarchical multi-agent optimization for resource allocation in cloud computing," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 3, pp. 692–707, 2021.

[8] D. Cui, Z. Peng, J. Xiong, B. Xu, and W. Lin, "A reinforcement learning-based mixed job scheduler scheme for grid or iaas cloud," *IEEE Transactions on Cloud Computing*, vol. 8, no. 4, pp. 1030–1039, 2020.

[9] S. S. Mondal, N. Sheoran, and S. Mitra, "Scheduling of time-varying workloads using reinforcement learning," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 9000–9008.

[10] H. Djigal, J. Feng, J. Lu, and J. Ge, "IPPTS: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 5, pp. 1057–1071, 2021.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nat.*, vol. 518, no. 7540, pp. 529–533, 2015.

[12] D. R. Karger, C. Stein, and J. Wein, "Scheduling algorithms," in *Algorithms and Theory of Computation Handbook*, ser. Chapman & Hall/CRC Applied Algorithms and Data Structures series, M. J. Atallah, Ed. CRC Press, 1999.

[13] F. Chen, M. S. Kodialam, and T. V. Lakshman, "Joint scheduling of processing and shuffle phases in mapreduce systems," in *Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012*, A. G. Greenberg and K. Sohraby, Eds. IEEE, 2012, pp. 1143–1151.

[14] K. Li, "Non-clairvoyant scheduling of independent parallel tasks on single and multiple multicore processors," *J. Parallel Distributed Comput.*, vol. 133, pp. 210–220, 2019.

[15] H. Xu, Y. Liu, and W. C. Lau, "Optimal job scheduling with resource packing for heterogeneous servers," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2021.

[16] A. Gupta, A. Kumar, and J. Li, "Non-preemptive flow-time minimization via rejections," in *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, ser. LIPIcs, I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, Eds., vol. 107. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 70:1–70:13.

[17] R. Ren, X. Tang, Y. Li, and W. Cai, "Competitiveness of dynamic bin packing for online cloud server allocation," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1324–1331, 2017.

[18] R. Dietze and G. Rünger, "The search-based scheduling algorithm hp* for parallel tasks on heterogeneous platforms," *Concurr. Comput. Pract. Exp.*, vol. 32, no. 21, 2020.

[19] Z. Wu and L. Fu, "Optimizing job completion time with fairness in large-scale data centers," *Future Gener. Comput. Syst.*, vol. 114, pp. 563–573, 2021.

[20] G. Utrera, M. Farreras, and J. Fornes, "Task packing: Efficient task scheduling in unbalanced parallel programs to maximize CPU utilization," *J. Parallel Distributed Comput.*, vol. 134, pp. 37–49, 2019.

[21] L. Shi, Z. Zhang, and T. G. Robertazzi, "Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in cloud," *IEEE Trans. Parallel Distributed Syst.*, vol. 28, no. 6, pp. 1607–1620, 2017.

[22] Z. Quan, Z.-J. Wang, T. Ye, and S. Guo, "Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1165–1182, 2020.

[23] D. S. Johnson, "Fast algorithms for bin packing," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 272–314, 1974.

[24] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 2000.

[25] D. J. Lehmann and M. B. Smyth, "Algebraic specification of data types: A synthetic approach," *Mathematical systems theory*, vol. 14, no. 1, pp. 97–139, 1981.

[26] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the European Conference on Computer Systems (EuroSys'15)*, Bordeaux, France, 2015.

[27] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: the Next Generation," in *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys'20)*. Heraklion, Greece: ACM, 2020.

[28] Y. Li, X. Tang, and W. Cai, "On dynamic bin packing for resource allocation in the cloud," in *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '14, Prague, Czech Republic - June 23 - 25, 2014*, G. E. Blelloch and P. Sanders, Eds. ACM, 2014, pp. 2–11.

[29] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, J. Wu and W. Hall, Eds. ACM, 2019, pp. 270–288.

[30] Y. Bao, Y. Peng, Y. Chen, and C. Wu, "Preemptive all-reduce scheduling for expediting distributed DNN training," in *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. IEEE, 2020, pp. 626–635.