IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 979-8-3503-8447-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/INFOCOMWKSHPS61880.2024.10620786

Online AI-Generated Content Request Scheduling with Deep Reinforcement Learning

Chenglong Feng, Ying Zheng, Yuedong Xu
Fudan University, Shanghai, China

Abstract—**Artificial Intelligence-Generated Content (AIGC) represents a technique that AI automatically generates various forms of content that meet the personalized requirements of humans. To offer an easy access to AIGC techniques, an edge computing system is introduced with heterogeneous computing capability to process users' text-to-image AIGC requests. For text-to-image AIGC tasks, inference steps configuration of diffusion models has a great impact on inference time and quality of generated results. Thus, it is necessary to design an online scheduling scheme to dispatch users' AIGC requests to suitable servers and tune proper inference steps for AIGC requests in order to achieve better quality of service. Based on this motivation, we define a novel online text-to-image AIGC request scheduling problem under a heterogeneous edge computing scenario with the objective of striking a balance between tardiness of AIGC requests and quality of generated results. We formulate it into an integer programming problem. Then we transform this problem into a Markov Decision Process (MDP) model and adopt a deep reinforcement learning-based algorithm to solve this online problem. Simulation results show that our approach performs better and has a strong generalization ability, compared with the baseline random policy and greedy policy.**

I. Introduction

In recent years, significant progress has been made in the field of Artificial Intelligence-Generated Content (AIGC). AIGC can make a huge contribution to our real life and society by promoting the efficiency and elevating the level of productivity, thanks to its strong ability to automatically and creatively generate various forms of content, such as text, images, audios [1], [2], videos [3] and so on. AIGC also impacts the world of arts to a large extent. Moreover, AIGC serves as a crucial technique to realize the concept of Metaverse [4].

In the domain of text generation AIGC, large language models, such as GPT [5] and LLaMA [6], exhibit outstanding performance and lead the burst of AIGC. Text generation AIGC techniques can be used as productive tools for question answering, article writing, language translation, coding, content summary and so on. For image generation AIGC like text-to-image and image-to-image tasks, diffusion-based models [7] such as stable diffusion [8] and DALL-E [9], [10] have evolved to be able to creatively generate high-quality images that exactly satisfy different input requirements. Image generation is of great practical value in image restoration [11], visual designing and so on. In the inference process of stable diffusion, a state-of-the-art diffusion model, prompt as well as other inference hyper-parameters need to be input to start the inference, such as inference steps, guidance scale and so on. In order to achieve the expected generated images within proper costs, inference hyper-parameters of diffusion models matter a lot and should be carefully designed besides the text prompt.

Despite the rapid development of AIGC, there arise several challenges to extensively apply AIGC techniques. First, both the training and inference of AIGC models require intensive computing resources, while users' local devices, such as mobile phones and head-mounted displays, usually can't meet the high hardware demand. Furthermore, AIGC models are large models that have billions of parameters. It turns out to be costly to deploy a great number of different large fine-tuned AIGC models and run the intensive computation on users' resource-constrained local devices. Considering these challenges, one feasible solution is to deploy AIGC models on a cluster of edge servers that have sufficient computing capability and capacity, and users can access specific AIGC services [12] through the edge network with flexibility and light burden on local devices. As AIGC tasks are offloaded to edge servers for processing, users only need to upload their AIGC requests, consisting of users' personalized prompts along with customized inference hyper-parameters. After the inference process of AIGC models is completed on edge servers, users can download corresponding generated content from the edge servers.

In this paper, we focus on a novel online scheduling problem that integrates key features of text-to-image AIGC tasks, under a heterogeneous edge computing scenario. For text-to-image AIGC tasks, the processing time and the results depend heavily on the setting of inference hyper-parameters. To provide the best performance of text-to-image AIGC services in an edge computing system, it is necessary to select the suitable servers for the AIGC requests uploaded by users and determine key inference hyper-parameters, such as inference steps. This scheduling problem is non-trivial for the following reasons. First, this scheduling problem is different from the classic job scheduling problem in that we take unique features of AIGC tasks into consideration. The processing time of each AIGC request is agnostic. The inference results are diverse with different configuration. Besides, we consider that AIGC requests arrive in an online fashion, so it is impossible to make a global optimal solution in practice. To tackle these challenges, deep reinforcement learning is applied and we design an Actor-Critic-based scheduling algorithm which suits the online AIGC request arriving scenario and has a strong generalization ability.

We first introduce a novel online AIGC request scheduling problem and formulate it into an integer programming problem, where the objective is to achieve better quality of AIGC results while optimizing the total tardiness. We reformulate it into an MDP model so that deep reinforcement learning can be applied. We carefully design the state and reward function, and adopt an Actor-Critic algorithm to solve this problem in an online fashion. Simulation results demonstrate that our algorithm performs better than the baseline random policy and greedy policy, and generalizes well to different scenarios. Our proposed AC-based scheduling algorithm improves the average quality of generated results by around 12% over the well-performed heuristic greedy policy and generally satisfy the average due time at the same time.

II. System Model

We introduce an edge computing system to process users' AIGC requests. We consider that the AIGC requests submitted by users are text-to-image generative tasks, which are executed by diffusion models, such as stable diffusion. Our edge computing system consists of a set of servers denoted as $\mathcal{J} = \{1, 2, 3, \cdots, J\}$, with heterogeneous capability c_j, representing the GPU compute capability of server $j \in \mathcal{J}$. More specifically, c_j indicates how much time it takes for server j to execute a single inference step of diffusion model. We assume that each server maintains homogeneous AIGC models. We denote $\mathcal{I} = \{1, 2, 3, \cdots, I\}$ as a set of AIGC requests submitted by users. AIGC requests arrive at our system in an online fashion with arbitrary time and order. In our request scheduling scenario, we consider that there is no preemption and each server can only execute one request at any time. Each server processes the requests one by one based on a First-Come-First-Serve (FCFS) strategy.

In order to efficiently and elastically process these AIGC requests, it is essential to design an intelligent online request scheduler for our edge computing system.

A. Scheduling Workflow

Users submit their AIGC requests to our edge computing system. We assume that requests are independent of each other. For each AIGC request $i \in \mathcal{I}$, the information of its arrival time r_i, due time d_i and tardiness penalty function $g_i(\cdot)$ is given by the user [13]. Due time d_i represents the preferred completion time of request i. Tardiness penalty function $g_i(\cdot)$, varying from request to request, specifies the latency cost. Specially for diffusion-based AIGC inference jobs, the execution time is relevant to the inference steps configuration. To make jobs elastic for scheduling and processing, inference steps configuration of each AIGC request will be considered as a decision variable instead of being set up directly by the users.

When an AIGC request arrives, we will select a proper server to process the inference job and also determine the inference steps configuration of it simultaneously. Once the inference steps configuration of request i on server j is determined, inference time can be estimated, denoted as p_{ij}. Then servers in parallel process requests. After the computation of jobs is completed, the corresponding generated content will be output and will be transmitted from the server back to the users immediately. When users receive the generated content, we can calculate the tardiness of request i, denoted as τ_i, and evaluate the quality of generated content, denoted as a function $Q(\cdot)$. These two components make up the utility function of our system.

As a result, our scheduler needs to determine two decision variables. We denote $x_{ijt} \in \{0, 1\}$ as a binary variable, indicating whether request i starts executing on server j at time t, and the inference steps decision variable of request i is denoted as s_i.

For the text-to-image AIGC requests, there exists a major trade-off between the inference time and quality of generated results. The purpose of our scheduler is to optimize the generated result of each AIGC request while minimizing the total tardiness to achieve the expected latency.

B. Inference Time Estimation

Unlike traditional job scheduling scenario, execution time of each job in our problem is agnostic before we make the dispatching decision. Inference steps and computational resources are two main factors that influence the inference time of a diffusion-based AIGC inference job. There exists an approximate linear relationship between the inference time of an AIGC job and its inference steps configuration with fixed GPU compute capability. Therefore, we can make an estimation of the inference time of each AIGC request, once the decision of request dispatching and inference steps has been made. The estimation can be represented as

$$p_{ij} = c_j s_i. \tag{1}$$

C. Quality of AIGC Results

There are several challenges in representing the quality of generated content in a mathematical way. First, AIGC models inherently introduce uncertainty and randomness. Moreover, the quality of generated results is subjective to users' perception and the same generated content may cause divergence among different users. There exist a general conclusion that the number of inference steps mainly influences the quality of text-to-image AIGC results [14]. We leverage a sigmoid function to represent the quality of AIGC results as

$$\begin{aligned} Q(s_i) &= 30 + 35 \times sigmoid(0.1 s_i - 9) \\ &= 30 + \frac{35}{1 + e^{-(0.1 s_i - 9)}}. \end{aligned} \tag{2}$$

Generally, more inference steps can bring about better AIGC results. Over-conservatively small inference steps configuration results in poor quality of AIGC results, defined as around 30, while over-aggressively large inference steps configuration leads to ceiling quality of AIGC results, defined as nearly 65. Inference steps configuration ranging from 30 to 150 has significant impacts on the quality of corresponding generated content [14].

D. Tardiness Penalty Function

Latency is another critical criterion in our system. If a request is completed over its due time, we need to impose a tardiness penalty to the objective function. We consider that various requests may exhibit different sensitivity to tardiness. Some requests are urgent, so we need to make sure that they are completed before due time, while certain tardiness is tolerated for other requests. We use three types of non-decreasing functions to represent tardiness penalty function [15], which are $g_{c_1}(t) = t$, $g_{c_2}(t) = t^2$ and $g_{c_3}(t) = \sqrt{t}$. Tardiness penalty function can be further defined as

$$g_i(t) = \begin{cases} g_c(t), & \text{if} t > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Different AIGC requests could have different tardiness penalty functions, indicating their tolerance of tardiness.

Important notations are listed in Table I.

TABLE I: Main Notations

Inputs	Descriptions
$\mathcal{I}$	the set of requests
$\mathcal{J}$	the set of servers
T	number of discrete time slots
r_i	arrival time of request i
d_i	due time of request i
$g_i(\cdot)$	tardiness penalty function of request i
c_j	GPU compute capability of server j
Decisions	Descriptions
x_{ijt}	whether request i starts(1) or not(0) at time t on server j
s_i	inference steps of request i

E. Problem Formulation

We formulate our online AIGC request scheduling problem into an integer programming problem. The objective of our optimization problem is to achieve the better quality of AIGC results while minimizing the total tardiness [16].

$$\min \sum_{i \in \mathcal{I}} g_i(\tau_i) - Q(s_i) \tag{4}$$

$$\text{s.t.} \sum_{j \in \mathcal{J}} (t + c_j s_i) x_{ijt} \le d_i + \tau_i, \ \forall i \in \mathcal{I}, t \in [T], \tag{4a}$$

$$\sum_{t \in [T]} \sum_{j \in \mathcal{J}} x_{ijt} = 1, \ \forall i \in \mathcal{I}, \tag{4b}$$

$$\sum_{i \in \mathcal{I}} \sum_{t' \in T_{ijt}} x_{ijt'} \le 1, \ \forall j \in \mathcal{J}, t \in [T], \tag{4c}$$

$$x_{ijt} = 0, \ \forall j \in \mathcal{J}, t < r_i \text{ or } t > T - c_j s_i, \tag{4d}$$

$$x_{ijt} \in \{0, 1\}, s_i \in Z^+, \ \forall i \in \mathcal{I}, j \in \mathcal{J}, t \in [T]. \tag{4e}$$

Constraint (4a) indicates that each request executes between its arrival time and due time with τ_i representing the tardiness of request i. Constraint (4b) ensures that each request starts once on exactly one server. Constraint (4c) ensures that each server can only execute one request at any time, which indicates that each server processes requests in a sequential way. $T_{ijt} = \{t' \mid t - c_j s_i < t' \le t\}$ is the set of discrete time slots at which request i in progress on server j at time t might start executing. Constraint (4d) is the time windows and ensures that each request starts only after its arrival time.

III. Algorithm Design

As our AIGC request scheduling problem is online, we attempt to solve this optimization problem with the help of deep reinforcement learning. More specifically, we apply an Actor-Critic (AC) algorithm to solve the online AIGC request scheduling problem.

A. Problem Transformation

We model the scheduling process as a Markov Decision Process (MDP). We first present the formulation of the MDP model.

1) State: At each decision step t, the agent observes the state of our scheduling environment, consisting of two components: current status of server cluster, and characteristics of the arriving request. We represent the state as a vector $s_t = (m_1, m_2, m_3, \cdots, m_J, o_t, o_p)$, where m_j is the current remaining time of all requests dispatched on server j, $o_t = d_i - r_i$ indicates the expected processing time of currently arriving request i, and $o_p = g_i(5)$ is the feature that specifies the sensitivity to tardiness of currently arriving request i.

2) Action: When an AIGC request arrives, we need to dispatch this request to a suitable server and determine its inference steps configuration. Once the state s_t is observed, the agent takes an action a_t, consisting of job dispatch decision $a^m \in \{1, 2, 3, \cdots, J\}$ and inference steps configuration decision $a^{steps} \in \{1, 2, 3, \cdots, 200\}$. We represent the action as a vector $a_t = (a^m, a^{steps})$. Therefore, the action space is a two-dimensional hybrid space with a hierarchical structure.

3) Reward: When the agent takes an action, current state transits to another state and an immediate reward is calculated to evaluate the action. In our request scheduling problem, the optimization goal is to minimize the tardiness as well as ensure the quality of AIGC results. The reward signal should be crafted to guide the agent towards good solutions to our optimization goal. As a result, we design the reward function r_t as

$$r_t = Q(a^{steps}) - w \cdot g_i(t^{fin} - d_i), \tag{5}$$

where w is a scaling factor and t^{fin} denotes the finish time of the currently arriving request i, which can be easily calculated when a decision action is taken. It is obvious that good quality of AIGC results will be positively rewarded, while violating the due time will be punished.

B. AC-based Scheduling Algorithm

We adopt an actor-critic-based deep reinforcement learning algorithm to solve the MDP problem above. AC framework combines the advantages of both value-based and policy-based reinforcement learning. The AC algorithm that we apply consists of an actor network and a critic network. The actor network is used to learn the policy $\pi_\theta(s_t, a_t)$, which is a probability that action a_t is taken in state s_t. The critic network is used for policy optimization and approximating the state-value function $V^{\pi}(s_t)$.

In our problem, the action space is hybrid, which includes two coupled action units. Therefore, we specifically design a two-branch-structured neural network as the actor network, which can output a two-dimensional result. For the actor neural network, one branch outputs a probability distribution as the policy of how to dispatch a request and the other branch outputs another probability distribution over all possible inference steps as the policy of how to determine the inference steps of a request. The critic network is a multi-layer perceptron to approximate state-value function $V^{\pi}(s_t)$, where we input state vector s_t and the estimated state-value $\hat{v}(s_t)$ can be output.

During the training, the parameters of the critic network are updated through TD error δ_t. TD error is computed as

$$\delta_t = r_t + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t), \tag{6}$$

where γ is the discount factor. We update the parameters of the actor network with the help of the policy gradient method and advantage function $A^{\pi}(s_t, a_t)$. The policy gradient is computed as

$$\nabla_{\theta} J(\theta) = E_t[\nabla_{\theta} \log \pi_{\theta}(s_t, a_t) A^{\pi_{\theta}}(s_t, a_t)], \tag{7}$$

where $\nabla_{\theta} J(\theta)$ denotes the policy gradient. The advantage function $A^{\pi}(s_t, a_t)$ is defined as

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t). \tag{8}$$

We can estimate the action-value $Q^{\pi}(s_t, a_t)$ as

$$Q^{\pi}(s_t, a_t) \approx r_t + \gamma V^{\pi}(s_{t+1}). \tag{9}$$

Therefore, in the algorithm, the advantage function is further computed as

$$A^{\pi}(s_t, a_t) = r_t + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t). \tag{10}$$

C. Training Process

We train the AC agent with various online AIGC request arrival scenarios in an episodic setting. Algorithm 1 demonstrates the training process of our AC-based scheduling algorithm. In the beginning of each episode, the simulation environment resets a new AIGC request arrival sequences for scheduling. At each decision step t in an episode, an AIGC request arrives. The AC agent observes the current state s_t of the environment, and then takes an action a_t based on the policy generated by the current actor network. After the action, the arriving AIGC request is scheduled, and reward r_t and next state s_{t+1} are obtained by the AC agent. The AC agent stores the tuple (s_t, a_t, r_t, s_{t+1}). When all AIGC requests have been scheduled, the episode terminates. At the end of each episode, the AC agent updates its critic network and actor network with the trajectories collected during the episode. Therefore, the AC agent learns in an episode-by-episode way.

IV. Performance Evaluation

In this section, we conduct simulation experiments and present the performance evaluation of our proposed AC algorithm for solving the online AIGC request scheduling problem.

Algorithm 1: AC-based Scheduling Algorithm

```
Input: AIGC request set $\mathcal{I}$, server set $\mathcal{J}$, scaling
       factor $w$, number of training episodes $E$, the
       learning rate, the discount factor $\gamma$.
Output: Optimized policy $\pi_\theta$.
1  Initialize actor network and critic network parameters;
2  for $e = 1, 2, \ldots, E$ do
3      The sever cluster is reset;
4      Generate a new AIGC request online arriving
        instance;
5      for $t = 1, 2, \ldots, I$ do
6          The agent observes current state $s_t$;
7          The agent chooses an action $a_t$ with current
            actor network based on $s_t$;
8          Dispatch the request to a server according to
            the action $a^m$;
9          Determine the inference steps configuration of
            the request according to the action $a^{steps}$;
10         Compute the reward $r_t$ according to (5);
11         The server cluster executes AIGC requests
            during the time interval and updates the status;
12         The environment transits to next state $s_{t+1}$;
13         The agent stores the transitions $(s_t, a_t, r_t, s_{t+1})$
            into the trajectory $\tau$;
14     end
15     Update the actor network and critic network with
        the trajectory $\tau$ according to (7) and (6);
16 end
```

A. Experimental Settings

In our simulation environment, we consider a cluster of 10 servers to process 2000 AIGC requests. Each server is initialized with a random GPU compute capability. Based on real-world practical experience, it takes around 3 seconds for one Tesla A100 GPU to finish a text-to-image AIGC inference task with 50 inference steps, which implies that we can define the GPU compute capability of Tesla A100 as about 17 inference steps per second. In this light, we simulate a heterogeneous server cluster as follows: 6 servers are equipped with stronger GPU compute capability c_j, ranging from 12 to 18, representing GPUs like Tesla V100 and A100, while the other 4 servers are equipped with relatively weak GPU compute capability c_j, ranging from 6 to 10, representing GPUs like NVIDIA GeForce RTX 3060. The arrival of AIGC requests is modeled as a Poisson process, where the average arrival rate λ is $\frac{1}{15}$. For each AIGC request, we randomly choose one tardiness penalty function. The due time of each AIGC request is generated after its arrival time with a random range of 2 to 20. The scaling factor w in the reward function is 1.5 to better balance the latency and quality of AIGC results.

B. Baseline

In our experiment, we compare our AC-based scheduling algorithm with some heuristic policies. The baseline heuristic policies can give a solution of scheduling but suffer from partiality in some ways.

1) Random Policy: In the random policy, we randomly select a server to process an arriving AIGC request and we only consider the quality of generated content by fixing a large inference steps configuration of each AIGC request as 120 without consideration of due time.

2) Greedy Policy: The greedy policy schedules the AIGC requests based on the principle that each AIGC request should start executing as soon as possible. We assign the arriving AIGC request to a server that has the shortest remaining time in the moment. As for the inference steps configuration of each AIGC request, the greedy policy will determine a larger inference steps for the AIGC request, under which the AIGC request can be completed exactly before the due time. Therefore, with the greedy policy, each AIGC request will not violate its due time. Although the greedy policy serves as a good heuristic solution to our scheduling problem, it performs poorly under some particular scenarios and it also neglects the trade-off between the latency and quality of results in our scheduling problem.

C. Experimental Results

1) Training: We train the AC agent for 3000 episodes. The results are presented in Fig. 1. As is shown in Fig. 1, the AC agent converges after around 1000 episodes of training. The AC agent gradually learns from various AIGC request arriving instances and develops a relatively optimal policy that can adapt to different scenarios. Compared with the random policy and the greedy policy, our proposed AC-based scheduling algorithm gains more rewards and performs better after it is well-trained.

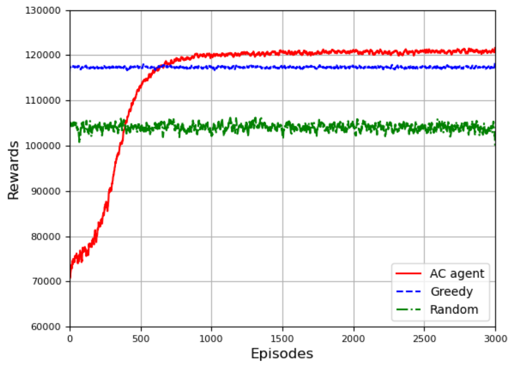
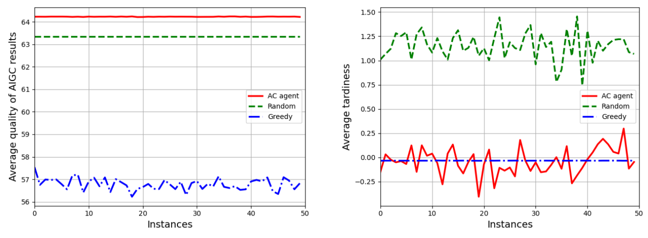


Fig. 1: Learning curve of the AC agent compared with baseline.

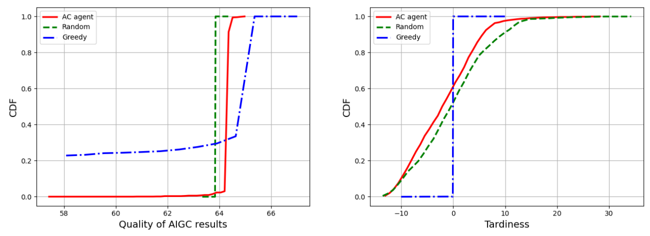
2) Testing: We test the performance of our AC agent with various simulation experiments. We mainly focus on two performance metrics in our scheduling problem, which is the average quality of generated content and average tardiness of AIGC requests. As is shown in Fig. 2, our proposed AC agent can achieve better AIGC results on average while minimizing the average tardiness, at 50 different AIGC request arriving instances. Fig. 2 demonstrates that the AC agent generalizes well to different instances. Fig. 3 shows the scheduling results and plots the performance distribution of the scheduled 2000 AIGC requests. Our AC agent balances the trade-off between the quality and the latency for each AIGC request by allowing slightly sacrificing the request completion time for better quality of generated content.

We further explore the scheduling performance at different simulation environment settings. We change the inter-arrival time, which is the reciprocal of arrival rate, and plot the scheduling performance in Fig. 4. Fig. 4 shows that our AC agent outperforms the greedy policy and random policy under different inter-arrival time. As the inter-arrival time of AIGC requests increases, the AC agent displays better performance in terms of reducing the average tardiness. We then test our trained AC agent on different server clusters. The server cluster is made up of 10 servers in total and we change the number of servers with stronger capability from 2, 4, 6 to 8. Fig. 5 demonstrates that the scheduling performance improves when the edge computing system has stronger computing resources. When the computing resources are relatively limited, it becomes necessary to apply the AC agent to make the scheduling decision.



(a) Average quality of AIGC results (b) Average tardiness

Fig. 2: Performance under 50 different request arriving instances. The AC agent shows a good generalization ability.

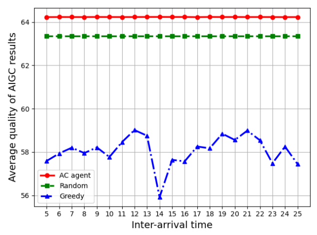


(a) Quality of AIGC generated content for each AIGC request (b) Tardiness of each AIGC request

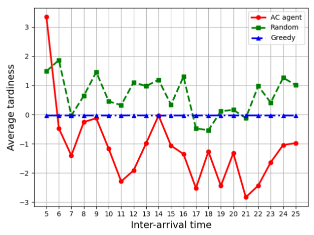
Fig. 3: Performance distribution of each scheduled AIGC request. The AC agent strikes a balance between the quality and latency.

D. Discussion about the Performance Gain

We try to interpret why the AC agent achieves better performance in our scheduling problem than the random policy and the greedy policy. First, as for the random policy, it completely neglects the information of arriving requests and the status of

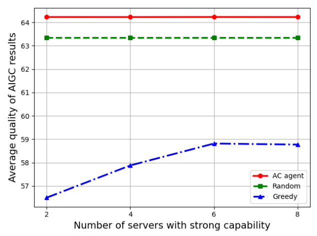


(a) Average quality of AIGC results

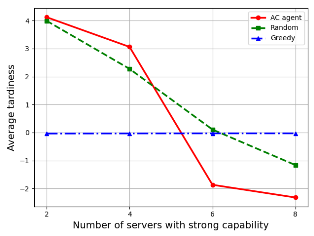


(b) Average tardiness

Fig. 4: Performance under different request inter-arrival time.



(a) Average quality of AIGC results



(b) Average tardiness

Fig. 5: Performance under server clusters with different computing resources.

servers at each decision point. So the random policy is very unstable and can only serve as a lower bound of scheduling performance. Next, the greedy policy, as a human-crafted heuristic rule, provides a better solution to our scheduling problem. Some important characteristics of our scheduling problem, such as the changing status of servers and the due time of each request, are taken into consideration. However, the greedy policy overlooks each request's different sensitivity to tardiness when making the inference steps configuration decision. Therefore, the greedy policy is still biased and partial.

Due to the strong learning ability, the AC agent can capture the key features in our scheduling problem and make the decision from the global viewpoint. The trained AC agent strikes a balance between tardiness and quality of AIGC results (Fig. 3). A certain degree of tardiness is allowed for the AC agent as long as the better quality of AIGC results can cover the penalty of tardiness. For the AIGC request that is sensitive to tardiness, the AC agent dispatches it to a server that has stronger capability and shorter remaining time, and determines moderate inference steps for it. In conclusion, our proposed AC-based scheduling algorithm improves the scheduling performance through its specific design towards our scheduling problem, strong learning ability and comprehensive consideration.

V. Conclusion

This work focuses on a novel online text-to-image AIGC request scheduling problem in an edge computing system with heterogeneous computing capability. We first present the formulation of our online AIGC request scheduling problem. Through experimental observations and empirical conclusions, we propose methods to estimate inference time of AIGC requests and evaluate the quality of AIGC generated results. To solve this problem in an online fashion, we apply deep reinforcement learning and design an AC-based scheduling algorithm. We conduct simulation experiments to evaluate the performance of our proposed AC-based scheduling algorithm against the baseline random policy and greedy policy. Experimental results show that our approach achieves more rewards and generalizes well to different instances due to its specific design and strong learning ability.

Acknowledgement

This work was supported in part by the Natural Science Foundation of China under Grant Grant 62072117, the Shanghai Natural Science Foundation under the Grant 22ZR1407000, and the Key-Area Research and Development Program of Guangdong Province under Grant 2020B010166003.

References

[1] C. Chen, Y. Hu, W. Weng, and E. S. Chng, "Metric-oriented speech enhancement using diffusion probabilistic model," in *IEEE ICASSP*, 2023, pp. 1–5.

[2] Z. Borsos, R. Marinier, and *et al.*, "Audiolm: A language modeling approach to audio generation," *IEEE/ACM Trans. Audio, Speech, and Language Processing*, vol. 31, pp. 2523–2533, 2022.

[3] Z. Luo, D. Chen, and *et al.*, "Videofusion: Decomposed diffusion models for high-quality video generation," *IEEE/CVF Conf. CVPR*, pp. 10 209–10 218, 2023.

[4] Y. Wang, Z. Su, N. Zhang, R. Xing, D. Liu, T. H. Luan, and X. Shen, "A survey on metaverse: Fundamentals, security, and privacy," *IEEE Communications Surveys Tutorials*, vol. 25, no. 1, pp. 319–352, 2023.

[5] L. Ouyang, J. Wu, and *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 27 730–27 744.

[6] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *ArXiv*, 2023.

[7] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 6840–6851.

[8] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *IEEE/CVF Conf. CVPR*, 2022, pp. 10 674–10 685.

[9] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," *ArXiv*, 2021.

[10] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *ArXiv*, 2022.

[11] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. V. Gool, "Repaint: Inpainting using denoising diffusion probabilistic models," *IEEE/CVF Conf. CVPR*, pp. 11 451–11 461, 2022.

[12] M. Xu, H. Du, D. Niyato, and *et al.*, "Unleashing the power of edge-cloud generative ai in mobile networks: A survey of aigc services," *IEEE Communications Surveys Tutorials*, vol. PP, pp. 1–1, 2024.

[13] R. Zhou, Z. Li, C. Wu, and Z. Huang, "An efficient cloud market mechanism for computing jobs with soft deadlines," *IEEE/ACM Trans. Networking*, vol. 25, no. 2, pp. 793–805, 2017.

[14] H. Du, Z. Li, D. Niyato, J. Kang, Z. Xiong, X. S. Shen, and D. I. Kim, "Enabling ai-generated content services in wireless edge networks," *IEEE Wireless Communications*, pp. 1–9, 2024.

[15] F. Wang, L. Jiao, K. Zhu, and L. Zhang, "Online edge computing demand response via deadline-aware v2g discharging auctions," *IEEE Trans. Mobile Computing*, vol. 22, no. 12, pp. 7279–7293, 2023.

[16] J. Hooker, "Planning and scheduling to minimize tardiness," in *International Conference on Principles and Practice of Constraint Programming*, 2005.