DBM: Delay-sensitive Buffering Mechanism for DNN Offloading Services

Guoliang Gao[1,2,3], Liantao Wu[1], Yang Yang[4,5], and Kai Li[1]

[1]School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China
[2]Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai 200050, China
[3]University of Chinese Academy of Sciences, Beijing 101408, China
{gaogl, wult, likai}@shanghaitech.edu.cn
[4]Terminus Group, Beijing 100027, China
[5]Peng Cheng Laboratory, Shenzhen 518055, China
dr.yangyang@terminusgroup.com

***Abstract*—DNN offloading has become an important supporting technology for edge intelligence. However, most of the existing works do not consider thread scheduling, which can achieve the parallelism of multiple threads in the practical distributed DNN inference system. To address this issue, we discuss the thread scheduling of the computing units participating in offloading in this paper, considering a single-core Central Processing Unit (CPU) and the Round Robin Scheduling (RRS). We deduce the relationship between the blocking of DNN inference-related threads and the Average Task Delay (ATD) and prove that an appropriate buffer setting can reduce blocking times. Theoretical analysis verifies that the buffering mechanism (DBM) can reduce the ATD significantly, and experimental results demonstrate that the DBM-improved DNN offloading can achieve a delay reduction of 14%-71%.**

***Index Terms*—DNN offloading, streaming tasks, buffering mechanism**

I. Introduction

Deep Neural Networks (DNNs) have pushed artificial intelligence to an unprecedented height [1]. Many applications have reached the precision level that can be practically applied, such as face recognition [2], video analysis [3], and natural language processing [4]. However, high-precision DNNs usually require high computing power [5]. They must run on high-performance processors, or the task delays will be intolerable [6].

In order to enable devices with low computing power, such as mobile phones and VR helmets, to benefit from deep learning, some works explore DNN offloading. They offload users' DNN tasks to computing units such as cloud and edge servers, which complete the execution of DNNs, and then return the results to users. Traditionally, users' DNN tasks were offloaded to a single unit [7], and the unit is usually the cloud [8], [9]. Now, partitioning DNNs into multiple parts and offloading them to multiple units is another mainstream [10].

So far, a widely studied category of the offloading model is the single-task model. It treats a DNN and its required input data as a whole, optimizing the total delay of the DNN processing all the data. The related work can be divided into three categories: offloading schemes for different device networks (edge [11]–[13], edge-cloud [14]–[16]), joint optimization of delay and other indicators (energy [17]–[19], privacy [20]), as well as the resource competition and task queuing process in the multi-task-multi-unit scenario [21], [22].

As a critical DNN task type, streaming tasks have gained increasing popularity [14], [23]. [14] initially introduced the pipeline idea, while [23] (OCDST) formally proposed the concept of streaming tasks. When performing streaming tasks, the involved computing units work as a pipeline. Parallelizing the steps in the pipeline can significantly reduce the delay. OCDST completes the transformation from theoretical research to practical implementation by mapping the DNN execution into multiple receivers (receiving threads), computers (computing threads), and publishers (publishing threads) [24]. In each unit, the receiving (producer) and computing (consumer) threads compose a producer-consumer pattern, and the computing (producer) and publishing (consumer) threads form another producer-consumer pattern [25]. However, we noticed that when analyzing the Average Task Delay (ATD) of a distributed DNN inference system implemented by the multithreading technique. OCDST only considered the executing delay of each thread, ignoring the delay introduced by thread scheduling. Therefore, the inferring speed of the practical system is slower than that derived from the theory. To address this issue, we analyze the thread scheduling process in OCDST based on a single-

This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB1803304; and in part by the National Natural Science Foundation of China under Grant 62202307.

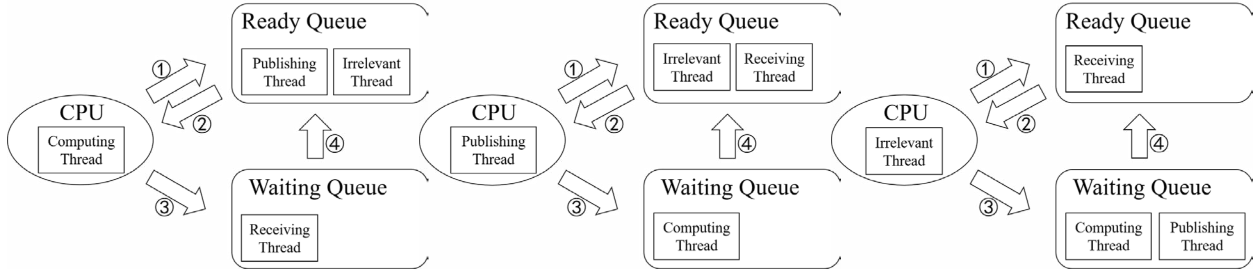

Fig. 1: Scheduling Status (1). Fig. 2: Scheduling Status (2). Fig. 3: Scheduling Status (3).

core Central Processing Unit (CPU) and the Round Robin Scheduling (RRS) and prove that the buffering mechanism can further reduce the ATD. The demonstration includes three steps: firstly, we show that an appropriate buffer setting reduces the blocking times of the producer thread to 0 or those of the consumer thread to 1. Next, we deduce the quantitative relationship between the blocking times and the ATD. Finally, we substitute the buffer setting into the quantitative relationship and obtain the reduced ATD value. We implement a distributed DNN inference system in which the receiving, processing, and publishing threads on each computing unit share a single-core CPU based on the RRS. We accelerate the inference by adequately setting the buffer size. The simulating results show that OCDST improved by the buffering mechanism achieves a delay reduction of 14%-71%. Our contributions are:

- Based on a single-core CPU and the RRS, the thread scheduling process in OCDST is analyzed, and a buffering mechanism is proposed to reduce the ATD further.
- A distributed DNN inference system is implemented, verifying the buffering mechanism's effectiveness numerically.

The rest of this paper is organized as follows. Section II analyses the thread scheduling process based on a single-core CPU and the RRS. Section III discusses the relevance between the thread blocking and the buffer size. Section IV deduces the relationship between the blocking of DNN inference-related threads and the ATD. Section V describes the experimentation. Finally, Section VI concludes this paper.

II. OCDST's Thread Scheduling

In a distributed DNN inference system handling streaming tasks, the ATD is mainly influenced by the task receiving, processing, and publishing delays of each computing unit, which are related to the size and the computing amount of the task, as well as the transmitting and computing capacity of the computing unit. OCDST has discussed the optimizing problem based on these three delays, minimizing the ATD by searching for the optimal offloading scheme. However, thread parallelism is usually implemented with multithreading, which introduces extra delays depending on the thread scheduling mode. OCDST assumes that each computing unit's receiving, computing, and publishing programs are entirely parallel without considering the impact of thread scheduling. Therefore, when implementing a practical system, the inference speed will be slightly slower than the theoretical value. Fortunately, the buffering mechanism can mitigate the impact of thread scheduling. Specifically, the task receiving, computing, and publishing threads on a unit behave as producers or consumers. The producer works only when the product queue (buffer) is not full, and the consumer works only when the product queue is not empty. Otherwise, the producer (consumer) threads will wait in the waiting queue. Besides, when the required conditions are met, the producer (consumer) threads in the waiting queue can not directly be executed but will be moved to the ready queue, waiting for the execution of the threads in front of it. The waiting time in the ready queue is the delay introduced by thread scheduling. Our idea is to prove that the buffering mechanism can decrease the waiting frequency of producer (consumer) threads, achieving a delay reduction. Next, we will elaborate on the thread scheduling process in the distributed DNN inferring system.

We discuss the thread scheduling process of a single-core CPU based on the RRS, where we set the same priority for all the threads. Each DNN inference-related thread needs to wait for two kinds of threads. One is the threads completing the inference together with it. For example, the computing thread must wait for the receiving and publishing threads. The other is the threads unrelated to DNN inference. Therefore, we assume that only four threads participate in the thread scheduling of the operating system – the receiving thread, the computing thread, the publishing thread, and the irrelevant thread. This assumption is only used to simplify the analysis and will not mislead the results. The time slice of the irrelevant thread is the sum of the time slices allocated to all the threads unrelated to DNN inference. Fig. 1 shows a possible thread scheduling status. Arrow ① represents the process of a thread transferring to a ready state from a running state. For a thread running on the CPU, if its time slice has been used up, the thread will be moved from the CPU to the ready queue, transferring to a ready state. Arrow ② represents the process of a thread transferring to a running state from a ready state. If no thread is executing

on the CPU, the scheduler will move the thread at the head of the ready queue to the CPU, and so the thread transfers to a running state. Arrow ③ represents the process of a thread transferring to a blocked state from a running state. For a thread running on the CPU, if it asks for some unavailable resources, it will transfer to a blocked state and sleep in the waiting queue until the resources are available. Arrow ④ represents the process of a thread transferring to a ready state from a blocked state. If the resources waited by a blocked thread become available, the thread will be awakened. As a result, it is moved to the ready queue to participate in a new round of scheduling.

In the thread scheduling status shown in Fig. 1, the computing thread is running on the CPU (running state), the publishing thread and the irrelevant thread are in the ready queue (ready state), and the receiving thread is in the waiting queue (blocked state). During the execution of the computing thread, it may process all the available data within a time slice so that it is blocked due to lack of data and moved to the waiting queue. The computing unit switches to the thread scheduling status shown in Fig. 2. The publishing thread at the head of the ready queue is scheduled to a running state, and the irrelevant thread advances one step in the ready queue. Since the computing thread has processed all the available data, the product buffer filled by the receiving thread becomes empty. Accordingly, the receiving thread transfers from a blocked state to a ready state, waiting for the CPU scheduling. We assume that the publishing thread also publishes all the available data within a time slice, and then it transfers to a blocked state. The corresponding thread scheduling status is shown in Fig. 3. The irrelevant thread is moved to the CPU, taking some operations irrelevant to the DNN inference. Its execution will not affect the states of receiving, computing, and publishing threads. We assume that the irrelevant thread cannot be blocked.

III. The Relation between Blocking and Buffering

In Section II, we enable each DNN inference thread running on the CPU to process all the available data within the allocated time slice and then be blocked. It highlights the role that buffer size plays in scheduling. If the buffer size is reasonable, it will reduce the possibility of the DNN inference threads transferring to a blocked state, which we will elaborate on below.

For a DNN inference system, the receiving (producer) and computing (consumer) threads on the same computing unit form a producer-consumer pattern, while the computing (producer) and publishing (consumer) threads form another producer-consumer pattern. The buffer we discuss is the corresponding product queue. Therefore, we only need to excavate the impact of the buffer size on producers and consumers, regardless of which thread is taking the role of a producer or consumer. We denote the producing speed of the producer as v_p and the consuming speed of the consumer as v_c. Then in a time slice τ, the producer can produce at most $n_p = v_p\tau$ products, and the consumer can consume at most $n_c = v_c\tau$ products. We set the buffer size to $\min(n_p, n_c)$. Accordingly, we have the following three conclusions:

- If $n_p > n_c$, the producer will be blocked in every scheduling round, and the consumer will be blocked at most once. Specifically, the producer fills the buffer with n_c products and then turns to a blocked state. The consumer spends the entire time slice consuming the n_c products. If the consumer runs before the producer at the first thread scheduling round, the consumer will be blocked once because the buffer is empty.
- If $n_p < n_c$, the producer will not be blocked, and the consumer will be blocked in every scheduling round. Specifically, the producer can only produce n_p products in a time slice. The producer will not be blocked since the buffer size is exactly n_p. However, the consuming capacity of the consumer is greater than the producing capacity of the producer, and thus the content in the buffer is not enough for the consumer to fully use the entire time slice. The consumer will be blocked after consuming all the products in the buffer.
- If $n_p = n_c$, the producer will not be blocked, and the consumer will be blocked at most once. In this case, the producer can spend a time slice filling the buffer, while the consumer needs a time slice to make the buffer empty. In the special case where the consumer runs before the producer at the first thread scheduling round, the consumer will be blocked as the first case.

Setting the buffer size n_b to $\min(n_p, n_c)$ is precisely the buffering mechanism proposed in this paper. If $n_b < \min(n_p, n_c)$, both the producer and consumer will be blocked after producing or consuming n_b products. They cannot make full use of their time slices. If $n_b > \min(n_p, n_c)$, there will be extra space with the size of $n_b - \min(n_p, n_c)$ in the buffer because each time, the producer (consumer) can only produce (consume) $\min(n_p, n_c)$ products. It leads to a waste of memory resources.

IV. Decreasing ATDs by Buffering Mechanism

With the buffer size $\min(n_p, n_c)$, we can reduce the blocking times of the producer to 0 or those of the consumer to at most 1. In other words, a suitable buffer setting can decrease the blocking times of DNN inference-related threads (these threads make up two producer-consumer patterns). Next, we demonstrate that the ATD decreases with blocking times dropping to prove that the buffering mechanism can reduce the ATD.

In a thread scheduling round, the theoretical maximum delay that a thread q can run on the CPU is the time slice allocated, denoted as τ_q. Thread q may transfer to a ready state or a blocked state after execution, waiting for

other threads to be scheduled, and then it will be scheduled again. We denote the sum of the executing time of other threads in one thread scheduling round as $\tau_{-\mathbf{q}}$. If q needs the CPU time t_q to accomplish its task, and q's utilizing rate towards the time slice is α_q, then the scheduling round amount z_q that q needs to participate in is

$$z_q = \frac{t_q}{\alpha_q \tau_q}. \tag{1}$$

In z_q thread scheduling rounds, the total waiting time t_q^w of the thread q is

$$t_q^w = z_q \tau_{-\mathbf{q}}. \tag{2}$$

Nevertheless, if q can fully use the time slice, it will only participate in $\alpha_q z_q$ thread scheduling rounds, and its waiting time will be $\alpha_q t_q^w$. Therefore, thread blocking increases the waiting delay of q by $(1-\alpha_q)t_q^w$. As a result, the user waiting time increases by

$$\delta t_q^w = (1-\alpha_q) t_q^w. \tag{3}$$

Recall that when $n_b < \min(n_p, n_c)$, the producer and consumer will be blocked in every scheduling round. We denote producer and consumer threads as q'. The analysis related to q' applies to both producers and consumers. Assuming that the number of tasks to be processed is n_a, and each task requires q' to run for t_0 on the CPU, then the total time $t_{q'}$ that q' needs to run on the CPU is

$$t_{q'} = t_0 n_a. \tag{4}$$

In each execution, q' can only process $\min(n_b, n_p, n_c)$ tasks. That is, the time $\tau'_{q'}$ that q' can run in one thread scheduling round is

$$\tau'_{q'} = t_0 \cdot \min(n_b, n_p, n_c). \tag{5}$$

The number of thread scheduling rounds ($z_{q'}$) that q' needs to participate in is

$$z_{q'} = \frac{t_{q'}}{\tau'_{q'}}. \tag{6}$$

Let the time slice allocating to q' be $\tau_{q'}$, then the utilization rate of q' to the time slice ($\alpha_{q'}$) is:

$$\alpha_{q'} = \frac{\tau'_{q'}}{\tau_{q'}} = \frac{t_0 \cdot \min(n_b, n_p, n_c)}{\tau_{q'}} \tag{7}$$

Denote the time that threads other than q' occupy the CPU in a scheduling round as $\tau_{-\mathbf{q}'}$, and then the time $t_{q'}^w$ that q' waiting for CPU scheduling is

$$t_{q'}^w = z_{q'} \tau_{-\mathbf{q}'}. \tag{8}$$

Accordingly, in the case where q' is blocked every time it runs, its delay will increase by $\delta t_{q'}^w$ compared with the case without blocking. The following formula figures out the value of $\delta t_{q'}^w$:

$$\begin{aligned}\delta t_{q'}^w &= (1-\alpha)t_{q'}^w \\ &= (1 - \frac{t_0 \cdot \min(n_b, n_p, n_c)}{\tau_{q'}}) \frac{t_0 n_a}{t_0 \cdot \min(n_b, n_p, n_c)} \tau_{-\mathbf{q}'} \\ &= (\frac{n_a}{\min(n_b, n_p, n_c)} - \frac{t_{q'}}{\tau_{q'}}) \tau_{-\mathbf{q}'}.\end{aligned} \tag{9}$$

Based on the analysis above, we can conclude about a computing unit's thread scheduling. Denote the bandwidth as v^t and the computing power of unit i as v_i^c. When processing one task, the computing amount, publishing, and receiving data sizes of unit i are c_i, d_i^p, and d_i^r, respectively. Then the task amounts that receiving, calculating, and publishing threads of unit i can deal with within a slice τ are $n_{i,rec} = \frac{v^t \tau}{d_i^r}$, $n_{i,com} = \frac{v_i^c \tau}{c_i}$, and $n_{i,pub} = \frac{v^t \tau}{d_i^p}$, respectively. Assuming that we set the buffer correctly and that the task amount is n_a, the three threads need to run on the CPU for the time of $t_{i,rec} = \frac{n_a}{n_{i,rec}}\tau$, $t_{i,com} = \frac{n_a}{n_{i,com}}\tau$, and $t_{i,pub} = \frac{n_a}{n_{i,pub}}\tau$, respectively. The number of scheduling rounds they participate in is $z_i = \frac{n_a}{\min(n_{i,rec}, n_{i,com}, n_{i,pub})}$. The time they wait for the irrelevant thread is $t'_i = z_i \tau'$, where τ' is the executing time of the irrelevant thread in a thread scheduling round. OCDST has explained that the ATD of a distributed DNN inference system is the longest single-task thread running time. Assuming that the thread with the longest running time is on the unit i, the ATD $\bar{t}$ is

$$\begin{aligned}\bar{t} =& \frac{t_{i,rec} + t_{i,com} + t_{i,pub} + t'_i}{n_a} \\ =& \tau(\frac{1}{n_{i,rec}} + \frac{1}{n_{i,com}} + \frac{1}{n_{i,pub}}) \\ &+ \frac{1}{\min(n_{i,rec}, n_{i,com}, n_{i,pub})}\tau'.\end{aligned} \tag{10}$$

The conclusions above are premised on the correct buffer size. Namely, $n_b = \min(n_{i,rec}, n_{i,com}, n_{i,pub})$. If $n_b < \min(n_{i,rec}, n_{i,com}, n_{i,pub})$, the number of scheduling rounds will be $z'_i = \frac{n_a}{n_b}$, and the ATD will increase by δt calculated as follows:

$$\delta t = \tau'(\frac{1}{n_b} - \frac{1}{\min(n_{i,rec}, n_{i,com}, n_{i,pub})}). \tag{11}$$

In conclusion, we theoretically deduce the ATD value δt reduced by the buffering mechanism.

V. Experimentation

A. Parameters

We implement a distributed DNN-inference simulating system with two single-core computing units. Initially, the computing power of each unit is 24.5 GFLOPS, and the bandwidth is 1 Gbps. The computing power is obtained from our Intel i7-8700 CPU with 12 cores. The tool "QwikMark" shows that its computing power is 294

GFLOPS. We use the single-core power of $294/12 = 24.5$. The bandwidth comes from our Gigabit LAN. Four threads are running on each computing unit: the receiving thread, which is responsible for receiving the input from the user or the intermediate output from the last unit; the computing thread, which is responsible for executing the DNN offloaded to the computing unit; the publishing thread, which is responsible for publishing the output to the next unit; irrelevant threads, which takes the operations unrelated to DNN inference. We set the time slice to be 0.1 s for all threads. It is the default slice of the Linux SCHED_RR Policy and can be checked by the command "sysctl kernel.sched_rr_timeslice_ms". The threads share the single-core CPUs on the computing units following the RRS. The streaming task here is an image stream with 1000 images. The image size is $224 \times 224 \times 3$. The involved DNNs are NiN [26], AlexNet [27] and ResNet-18 [5].

B. The Performance of OCDST with Buffering Mechanism

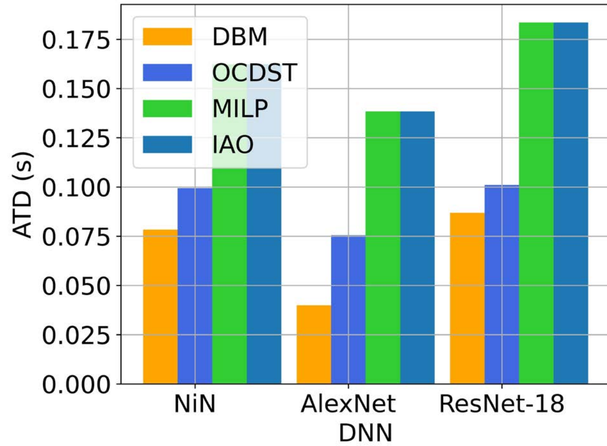

Fig. 4: The Performance of OCDST with Buffering Mechanism.

We compare the performance of the OCDST improved by the buffering mechanism (DBM) with the original OCDST [23], MILP [12], and IAO [21] methods. The metric is the inference ATD. The lower the ATD, the better the method. As shown in Fig. 4, DBM reduces the ATDs to 29%-86% of the others, while its ATDs are 53%-86% of those of OCDST.

C. Effects of System Parameters on DBM-enabled Offloading Services

We have explained that setting the buffer size of unit i to $\min(n_{i,rec}, n_{i,com}, n_{i,pub})$ can minimize the number of threads blocked by the buffer. According to the theoretical analysis in section IV, $n_{i,rec}$, $n_{i,com}$, and $n_{i,pub}$ are positively correlated with the time slice. Besides, $n_{i,rec}$ and $n_{i,pub}$ are positively correlated with the bandwidth, and $n_{i,com}$ is positively correlated with the computing power. In this section, we numerically verify the impact of system parameters on the buffer setting. Only one variable

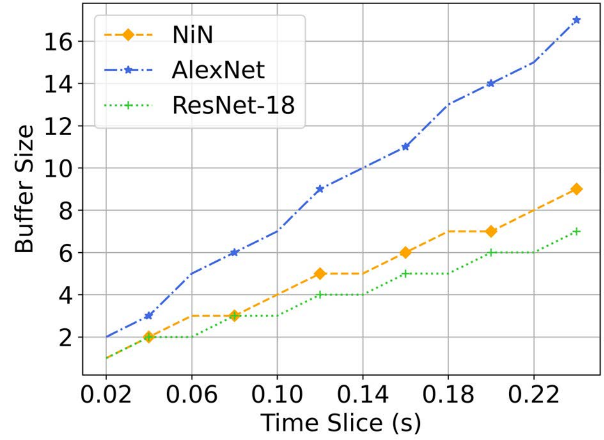

Fig. 5: Impact of Time Slice for DBM-enabled Offloading Services.

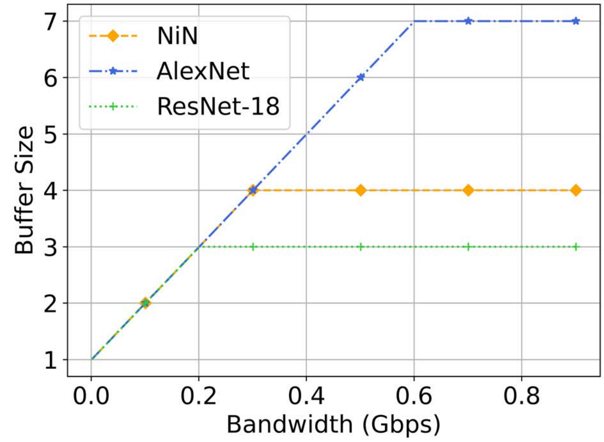

Fig. 6: Impact of Bandwidth for DBM-enabled Offloading Services.

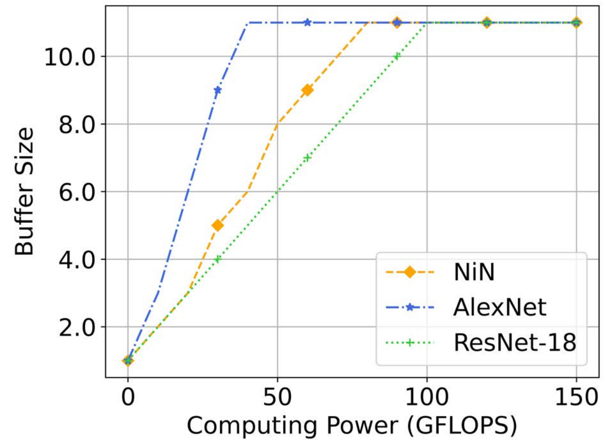

Fig. 7: Impact of Computing Power for DBM-enabled Offloading Services.

is changed in each experiment, and the other variables remain at their initial values. The results are shown in Fig. 5, 6, and 7. The tested offloading model is DBM.

Fig. 5 illustrates the impact of the time slice of DNN

inference-related threads on the buffer setting. It can be seen that as the time slice increases, the optimal buffer size keeps rising. A larger time slice allows threads to process more data in a scheduling round. Therefore, a larger buffer is required to reduce blocking times.

Fig. 6 presents the effect of bandwidth on the buffer settings. The optimal buffer size rises as the bandwidth increases. However, such a trend stops when the bandwidth exceeds a certain value. Increasing the bandwidth can raise the number of tasks received and published in a time slice. The uptrend reflects it. Nevertheless, the computing thread becomes the speed bottleneck in the case where the bandwidth is high enough. Since increasing the bandwidth will not speed up the execution of the computing thread, the optimal buffer size stops rising.

Fig. 7 shows the effect of computing power on the buffer setting. The trend is the same as in Fig. 6, and the causes are similar. Higher computing power enables the computing thread to process more tasks in a time slice, and thus the optimal buffer size rises. However, when the computing power is large enough, the transmitting threads (receiving and publishing threads) become the speed bottleneck. Increasing the computing power will not speed up the execution of the transmitting threads, and thus the optimal buffer size does not increase.

VI. Conclusion

In this paper, we study the thread scheduling process of DNN inference-related threads. The relationship between the thread blocking and the ATD is deduced. A buffering mechanism that can reduce the ATD by decreasing the blocking times is proposed. Experimental results show that the proposed buffering mechanism reduces the ATD to 29%-86% compared with the recent works.

References

[1] Y. Li, F. Sun, W. Song, Y. Wen, K. Li, J. Wang, and Y. Yang, "Retrospective thinking based multi-agent system for wireless video transmissions," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.

[2] R. He, J. Cao, L. Song, Z. Sun, and T. Tan, "Adversarial cross-spectral face completion for nir-vis face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 5, pp. 1025–1037, 2020.

[3] X. Wei, C. Yang, and S. Han, "Prediction, communication, and computing duration optimization for vr video streaming," *IEEE Transactions on Communications*, vol. 69, no. 3, pp. 1947–1959, 2021.

[4] R. Vyas, K. Joshi, H. Sutar, and T. P. Nagarhalli, "Real time machine translation system for english to indian language," in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2020, pp. 838–842.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[7] G. Yoon, G.-Y. Kim, H. Yoo, S. C. Kim, and R. Kim, "Implementing practical dnn-based object detection offloading decision for maximizing detection performance of mobile edge devices," *IEEE Access*, vol. 9, pp. 140 199–140 211, 2021.

[8] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, 2019.

[9] Y. Yang, "Multi-tier computing networks for intelligent iot," *Nature Electronics*, vol. 2, no. 1, pp. 4–5, 2019.

[10] M. Xue, H. Wu, and R. Li, "Dnn migration in iots: Emerging technologies, current challenges and open research directions," *IEEE Consumer Electronics Magazine*, pp. 1–1, 2022.

[11] Y. Chang, X. Huang, Z. Shao, and Y. Yang, "An efficient distributed deep learning framework for fog-based iot systems," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[12] M. Gao, W. Cui, D. Gao, R. Shen, J. Li, and Y. Zhou, "Deep neural network task partitioning and offloading for mobile edge computing," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[13] H. Wang, G. Cai, Z. Huang, and F. Dong, "Adda: Adaptive distributed dnn inference acceleration in edge computing environment," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, 2019, pp. 438–445.

[14] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1423–1431.

[15] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 854–863.

[16] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven off-loading for dnn-based applications over cloud, edge, and end devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5456–5466, 2020.

[17] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 683–697, 2022.

[18] M. Xue, H. Wu, G. Peng, and K. Wolter, "Ddpqn: An efficient dnn offloading strategy in local-edge-cloud collaborative environments," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 640–655, 2022.

[19] L. Wu, Z. Liu, P. Sun, H. Chen, K. Wang, Y. Zuo, and Y. Yang, "Dot: Decentralized offloading of tasks in ofdma based heterogeneous computing networks," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[20] C. Shi, L. Chen, C. Shen, L. Song, and J. Xu, "Privacy-aware edge computing based on adaptive dnn partitioning," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[21] X. Tang, X. Chen, L. Zeng, S. Yu, and L. Chen, "Joint multi-user dnn partitioning and computational resource allocation for collaborative edge intelligence," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[22] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint dnn partition deployment and resource allocation for delay-sensitive deep learning inference in iot," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9241–9254, 2020.

[23] G. Gao, L. Wu, Z. Shao, Y. Yang, and Z. Lin, "Ocdst: Offloading chained dnns for streaming tasks," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6.

[24] Y. Cai, C. Jia, S. Wu, K. Zhai, and W. K. Chan, "Asn: A dynamic barrier-based approach to confirmation of deadlocks from warnings for large-scale multithreaded programs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 1, pp. 13–23, 2015.

[25] R. Medhat, B. Bonakdarpour, and S. Fischmeister, "Energy-efficient multiple producer-consumer," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 560–574, 2019.

[26] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.