MagicBatch: An Energy-aware Scheduling Framework for DNN Inference on Heterogeneous Edge Servers in Space-air-ground Computation

Di Liu[1,2], Zimo Ma[1], Aolin Zhang[1], and Kuangyu Zheng[1,2](✉)

[1] Department of Mathematics and Theories,
Peng Cheng Laboratory, Shenzhen, China
[2] School of Electronic and Information Engineering,
Beihang University, Beijing, China
{diliu,mazimo,aolin2000,zhengky}@buaa.edu.cn

Abstract. With the fast development of space-air-ground computing scenarios, large UAVs, airships or HAPS (high altitude platform station), and satellites, are in the trend to have more powerful computation resources (e.g., heterogeneous types of GPUs), and can act as edge servers in the air. They are increasingly used for a large number of deep neural networks (DNN) inference applications, such as disaster monitoring, remote sensing, and agriculture inspection. However, these edge servers in the air always have a very limited energy supply. Thus, how to reduce their energy consumption to extend their working hours, while meeting the delay requirements of DNN inference tasks becomes a very important demand.

In this paper, we propose MagicBatch, an energy-aware scheduling framework for DNN inference workloads on edge servers (with heterogeneous GPUs) in the air. MagicBatch is based on our key finding, that various GPUs can have different energy and latency performance under different DNN inference batch sizes. Thus, MagicBatch is designed in two phases: In the offline analysis phase, it analyzes the execution latency and energy consumption performance of different DNN inference tasks on heterogeneous GPUs; In the online scheduling phase, we propose a heuristic energy-aware scheduling algorithm (PSO-GA) to better allocate heterogeneous GPU computing resources to various inference tasks. Evaluation on our emulation testbed shows that MagicBatch can achieve more than 31.3 % energy savings and 41.1 % throughput improvement compared with the state-of-the-art methods.

Keywords: DNN inference · Heterogeneous GPUs · Energy-aware scheduling · Space-air-ground computation · Edge computing

Prof. Kuangyu Zheng is the Corresponding Author.

1 Introduction

Witnessed the emerging Space Internet like Starlink, OneWeb, together with the increasing coverage and computing demands at remote areas, the space-air-ground networks have received more attention. With the development of deep learning technology, a large number of DNN inference applications, such as disaster (e.g., flood, fire) monitoring, remote sensing, and agriculture inspection, etc., are deployed at the edge of the network. Most of them are computation-intensive and require sub-second-level short latency. Directly using the DNN models on mobile devices may fail to meet the latency requirements. To alleviate the latency bottleneck, a better solution is to use the emerging edge computing paradigm [3]. Large unmanned aerial vehicles (UAVs), airships, HAPS (High Altitude Platform Station), or satellites are having increasingly improved heterogeneous computation resources (e.g., different CPUs, GPUs, FPGAs, ASICs), and can be used as edge servers to provide extensive computing support for various DNN applications. However, for either UAVs, HARPs, or satellites, due to their very limited battery capacities, they are usually energy constrained.

For edge servers, the deployment of multiple heterogeneous GPUs are emerging, and is becoming a possible future trend. On one hand, new generations of GPU innovated quickly, and common edge servers always equipped with different generations and types of GPUs. On the other hand, parallel computing capabilities enable outdated types of GPUs to still speed up DNN inference tasks compared with mobile devices, and even have the advantage of cost-effectiveness. Thus, the heterogeneous characteristics of GPU clusters on edge servers are not unusual [5].

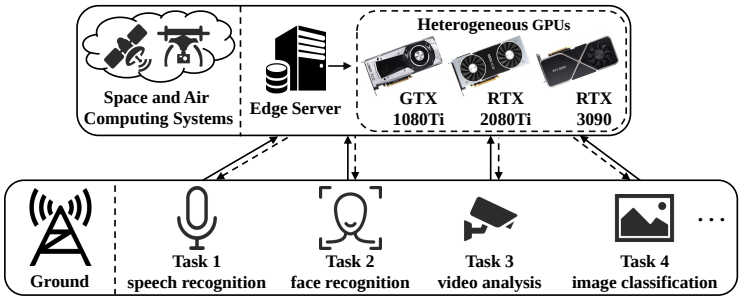


Fig. 1: Scenario: Ground devices submit various DNN tasks to the edge server in space or air with heterogeneous GPUs. (GPU types as GTX 1080Ti, RTX 2080Ti and RTX 3090, are merely heterogeneous examples we used in our emulation experiments)

Figure 1 shows a proposed working scenario of the edge servers in space-air-ground computation scenario. Ground devices send DNN inference requests to the edge server, and the edge server returns the results after completing the inference tasks on the heterogeneous GPUs. In order to make full use of parallel computing resources of GPUs, multiple DNN inference requests are usually

combined into a batch for execution. Higher system throughput and GPU resource utilization can be achieved by increasing the batch size and processing more inference tasks in parallel. However, a large batch size results in a longer execution latency, which may violate the deadline of the tasks. Therefore, the scheduler needs to consider appropriate batch size selection based on the trade-off between throughput and excution latency. Batch size also affects the energy consumption of GPUs when executing DNN inference tasks. Unfortunately, existing works have not paid attention to the potential of scheduling various DNN inference tasks on heterogeneous GPUs to save energy.

In this paper, we propose MagicBatch, an energy-aware scheduling framework for DNN inference workloads on heterogeneous edge servers in space-air-ground computing scenarios. We first analyze the inference performance of several classical DNN models on heterogeneous GPUs in detail. Then, a heuristic scheduling algorithm PSO-GA is designed to assign heterogeneous GPUs to various DNN inference tasks, thereby saving total energy consumption. Specifically, the contributions of this paper are as follows:

- We test and analyze the performance of various DNN inference tasks on some heterogeneous GPUs in detail, which confirm a key finding, that different GPUs could have different energy efficiencies and execution latencies, even running the same computation task. The analysis results are also used to predict the execution latency and energy consumption.
- We propose MagicBatch, a DNN inference tasks scheduling framework for heterogeneous edge servers. It leverages an energy-aware heuristic scheduling algorithm PSO-GA to reduce the energy consumption of edge servers while meeting the latency requirements of inference tasks.
- We evaluate the performance of MagicBatch through hardware experiments. The experimental results show that MagicBatch can significantly reduce the energy consumption of edge servers and improve the system throughput.

The rest of this paper is organized as follows. Section 2 surveys the related work. Section 3 identifies the latency and energy consumption performance characteristics of batching inference. Section 4 shows the design of MagicBatch. The performance evaluation of MagicBatch is conducted in Section 5. Finally, conclusions of this paper are provided in Section 6.

2 Related Work

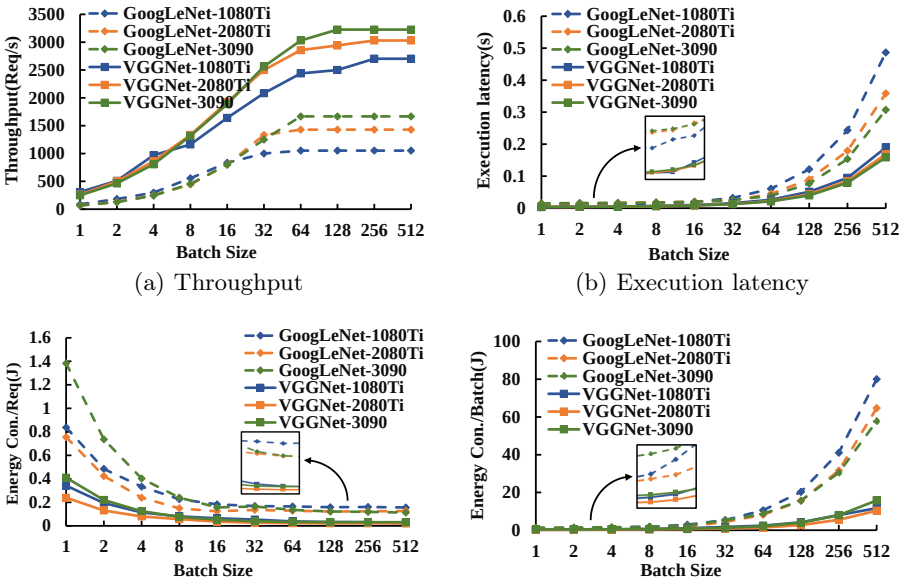
Some existing works focus on the heterogeneity of GPU clusters [6,9,11]. HetPipe [11] can effectively train large DNN models by using virtual workers composed of multiple heterogeneous GPUs. Gavel [9] is a heterogeneous-aware GPU cluster scheduler, which supports higher average input job rates and lower average job completion time. BytePS [6] is a unified distributed DNN training acceleration system that achieves optimal communication efficiency in heterogeneous clusters. However, these works focus on DNN training tasks rather than inference tasks with higher latency requirements.

Some schedulers aim at balancing throughput and QoS [2,4] in DNN inference workloads. Ebird [2] enables the GPU side prefetching mechanism and elastic batch scheduling strategy to improve the responsiveness and throughput of deep learning services. Kalmia [4] adopts a preemption-based scheduling method to reduce the deadline violation rate of DNN inference tasks with heterogeneous QoS requirements and achieve high system throughput. However, these works do not consider the optimization of energy consumption.

The energy consumption of GPUs is considered in some studies [8,12]. BatchDVFS [8] combines batch size selection with DVFS to maximize throughput while meeting the power cap. EAIS [12] adaptively coordinates batching processing and DVFS according to fluctuating workloads to minimize energy consumption and meet the latency SLO. However, the heterogeneity of GPUs in edge servers has not been fully considered in these works.

3 Batch Performance Analysis

In this section, we introduce the motivation experiment, by measuring the throughput, execution latency, and energy consumption of three heterogeneous GPUs when they execute inference tasks of two DNN models in different batch sizes.



(a) Throughput (b) Execution latency

(c) Energy consumption per request (d) Energy consumption per batch

Fig. 2: Throughput, execution latency, and energy consumption curves of heterogeneous GPUs executing inference tasks of diverse DNN models.

Throughput: Figure 2(a) shows the relationship between throughput and batch size. It can be seen that the throughputs of all these GPUs with different CNN tasks fluctuate at a high level after a sharp increase when the batch size

rises from 1 to 64. This is because the computing resources of GPUs have not been fully utilized when the batch size is small. At this stage, increasing the batch size enables the GPUs to execute more DNN inference tasks in parallel. When the workload of a batch exceeds the parallel processing capacity of the GPUs, the increase of the batch size cannot contribute to a higher throughput. Besides, we also find that GPUs with advanced architectures do not always perform better when executing different DNN inference tasks.

Execution latency: Figure 2(b) indicates the variation of execution latency with batch size. Specifically, execution time is positively correlated with batch size. The reason is simple. Although the batching strategy enables the GPUs to process more inference tasks in parallel at a time, the increase in batch size leads to longer execution time for individual batches, resulting in higher latency of requests. Thus, larger batch sizes imply higher system throughput, while smaller batch sizes imply higher task responsiveness.

Energy consumption: Figure 2(c) and 2(d) show the variation of energy consumption with batch size. It can be seen from 2(c) that the energy consumption for processing a single DNN inference task decreases with the increase of batch size, because parallel processing improves the effectiveness of GPU computing resources. Although a larger batch size may lead to higher execution power, the overall time to complete the same number of inference tasks decreases, so energy consumption can be reduced. Moreover, for both DNN tasks (VGGnet or GooglNet) as batch size increases, there are some crossing points (e.g., batch size 128 and 256) on the energy consumption between different types of GPUs (e.g., GPU 2080Ti and 3090), which means higher-end GPU (e.g. 3090) does not always have worse energy efficiency than lower-end ones (e.g. 2080Ti).

From the above analysis, it can be seen that heterogeneous GPUs have different characteristics in terms of throughput, latency, and energy consumption when executing DNN inference tasks. Besides, the performance of advanced GPUs is not always optimal. In this case, there is a lot of room for optimization in terms of energy consumption when scheduling different DNN inference tasks on heterogeneous GPUs.

4 MagicBatch Design

4.1 Overview

MagicBatch is an energy-aware scheduling framework for DNN inference tasks deployed on edge servers with heterogeneous GPU computing resources. Mobile devices continuously send inference requests to the edge server, while the server invokes MagicBatch to schedule the inference tasks to the appropriate GPU for execution, and sends the results back to each device. Figure 3 shows the overview of MagicBatch framework, which mainly consists of an offline analysis phase and an online scheduling phase.

In the offline analysis phase, MagicBatch measures and analyzes the performance of executing different inference tasks on multiple heterogeneous GPUs in

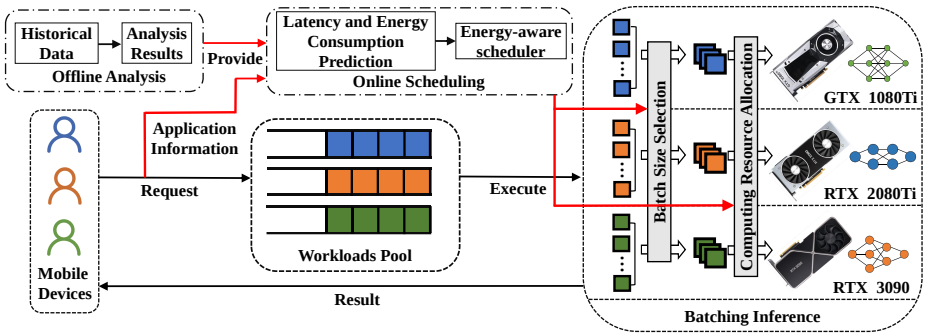


Fig. 3: Overview of MagicBatch framework.

different batch sizes, including throughput, execution latency, and energy consumption. The analysis results obtained in the offline analysis phase will be used in the online scheduling phase.

In the online scheduling phase, MagicBatch obtains application information of mobile devices (including the type of DNN models, task arrival rate, and task deadline) and assigns a task waiting queue for each device in the workloads pool. The inference requests of each device are added to their respective task queues and wait for execution. Then, MagicBatch predicts the latency and energy consumption for executing inference tasks in different batch sizes based on the results obtained from offline analysis phase (4.2). A heuristic scheduling algorithm is designed to select batch size and allocate computing resources to minimize energy consumption while meeting delay requirements of tasks (4.3).

4.2 Performance prediction

We predict the latency and energy consumption of inference tasks according to the anslysis results obtained in the offline analysis phase.

In the MagicBatch framework proposed in this paper, the inference requests of each device are first added to their respective task queues according to the order of arrival and then executed on the assigned GPU. Therefore, the latency of completing an inference request consists of two parts, the time spent waiting in the task queue and the time spent executing on the GPUs. Let $T_{request}$ represent the total latency, it can be expressed as:

$$T_{request} = T_{wait} + T_{execution} \tag{1}$$

Where T_{wait} denotes the waiting latency. $T_{execution}$ can be obtained from the offline analysis phase.

The time to wait in the task queue consists of two parts: the time to wait for subsequent requests and the time to wait for the assigned GPU to be idle. Figure 4 shows the complete process of an inference request. Suppose that at t_1, a new inference request is added to the task queue. In order to implement batching inference strategy, the request needs to wait for later requests. At t_2, the number of tasks reaches the pre-setting batch size, these tasks are combined

into a batch. However, the new batch cannot be executed immediately, because the GPU allocated to it may be busy at this time (such as t_3). Until t_4, the GPU finishes processing the last batch. The new batch of tasks finishes waiting and starts to be executed. It should be noted that in practice, the processing speed of GPU devices is much greater than the task arrival rate of inference requests from mobile devices. Therefore, without loss of generality, the latency of waiting for the GPU to be idle is negligible.

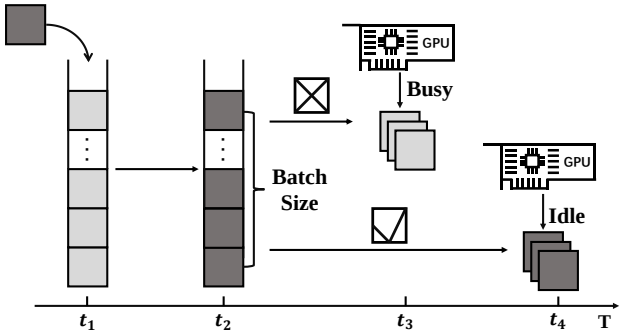


Fig. 4: The whole execution process of an inference request.

For each inference request in the same batch, the waiting time in the task queue is different, and the computation time on the GPU is the same. Therefore, as long as the first request in a batch does not exceed the upper latency limit, all tasks in the batch can be considered to meet the latency requirement. Let T_{wait}^{first} denotes the waiting time of the first arriving task, it can be expressed as:

$$T_{wait}^{first} = \frac{(B-1)}{r} \tag{2}$$

Where r represents the task arrival rate.

All requests in a batch are executed in parallel on the GPU, so the energy consumption can be regarded as equal. Let $E_{request}$ represents the energy consumption for a request:

$$E_{request} = \frac{E_{batch}}{B} \tag{3}$$

Where E_{batch} can be obtained from the offline analysis phase.

4.3 Scheduling algorithm

We first describe the optimization problem to be solved, and then design a heuristic algorithm PSO-GA to find an approximate optimal solution to the problem.

(1) Problem formulation.

Assume that the GPU cluster of the edge server consists of N types of heterogeneous GPUs, and the number of GPUs of each type is G_n $(1 \leqslant n \leqslant N)$. The current number of mobile devices sending inference requests to the edge server

is M, each device's requests are executed separately by the same GPU. The inference tasks of device m ($1 \leqslant m \leqslant M$) have the same task arrival rate r_m and deadline D_m. δ_n^m represents whether the inference tasks of m are executed on the n-type GPUs. If yes, $\delta_n^m = 1$ then otherwise $\delta_n^m = 0$. b_n^m denotes the optimal batch size for the inference tasks of m to process on the n-type GPUs. Without loss of generality, within a fixed duration τ, the system can be considered stable.

As can be seen from the previous discussion, execution latency and throughput increase monotonically with increasing batch size, while energy consumption does the opposite. Therefore, choosing the largest batch size that satisfies the latency requirement can guarantee minimum energy consumption at this time. The latency constraint can be expressed as:

$$T_{request}^{n,m} \leqslant D_m \tag{4}$$

Where $T_{request}^{n,m}$ represents the latency of executing the inference task of device m on n-type GPUs.

From (4), we can get the optimal batch size b_n^m for the inference tasks of device m on n-type GPUs. Then, according to the offline analysis results, we can obtain the energy consumption of a single task when executing the tasks of device m on n-type GPUs under the batch size b_n^m, expressed in $E_{request}^{n,m}$. In a duration τ, the GPUs' total inference energy consumption E_τ is shown in (5):

$$E_\tau = \sum_{m=1}^{M} \sum_{n=1}^{N} \delta_n^m \cdot E_{request}^{n,m} \cdot r_m \cdot \tau \tag{5}$$

Now we can get an optimization problem (P) as the following form:

$$(P) : minE_\tau \tag{6}$$

$$s.t. : T_{request}^{n,m} \leqslant D_m \tag{7}$$

$$\sum_{n=1}^{N} \sum_{g=1}^{G_n} \delta_n^m = 1, \sum_{m=1}^{M} \delta_n^m \leqslant G_n \tag{8}$$

$$M \leqslant \sum_{n=1}^{N} G_n \tag{9}$$

$$1 \leqslant m \leqslant M, 1 \leqslant n \leqslant N \tag{10}$$

where constraint (6) represents minimizing the sum of the energy consumption of all GPUs on the edge server. Constraint (7) means that no matter which GPU the inference task is executed on, the deadline cannot be exceeded. Constraint (8) means that the tasks of each device can only be executed on one GPU, and the number of GPUs allocated cannot exceed the total number of GPUs of this type. Constraint (9) means that the number of mobile devices that the server can accept cannot exceed the total number of GPUs.

For this optimization problem, the complexity of solving this problem increases rapidly as the number of servers accepting tasks increases. Meanwhile,

the utility function (6) is a nonlinear function, so it is essentially an NP-hard nonlinear integer optimization problem. Therefore, we design a heuristic algorithm that can provide a low-complexity near-optimal solution to this problem.

(2) PSO-GA algorithm.

PSO algorithm and GA algorithm are two classical optimization algorithms. PSO algorithm is easy to fall into local optimum, while GA algorithm has poor local search ability, resulting in low search efficiency in the later stage. Therefore, we combine PSO with GA to find the optimal scheduling strategy from a global perspective. Specifically, we use a particle to represent a scheduling strategy, and then iterate through the update strategies of PSO and GA, respectively, until an approximate optimal solution is found.

In this paper, GPU scheduling is a discrete problem, so we need to design a suitable coding scheme to improve the search efficiency and the performance of the PSO-GA optimization algorithm. Inspired by [7], we adopt a device-GPU nesting strategy for coding. Each particle represents a candidate scheduling scheme. The $i-th$ particle can be represented as follows:

$$X_i = (x_{i1}, x_{i2}, ..., x_{iM}) \tag{11}$$

Where x_{im} denotes the tasks of device m are executed by an n-type GPU, which can be expressed as follows:

$$x_{im} = (n, m)_{im} \tag{12}$$

After getting the coding results, we design fitness functions to evaluate the performance of particles. MagicBatch aims at minimizing the energy consumption of GPUs on the edge server, so we directly use the energy consumption model as the fitness function of PSO-GA algorithm. It should be noted that the number of GPUs allocated cannot exceed the total number of such GPUs, so not every particle can represent a feasible solution in the problem space. Therefore, We first divide the particles into two categories according to whether the restrictions are met, and then design a fitness function for each category of particles. When the number of GPUs allocated does not exceed the total number of such GPUs, fitness is the energy consumption value of the scheduling method represented by the particle. Instead, the fitness value is set to infinity. The final function expression is shown in (13):

$$F(X_i) = \begin{cases} E^{\tau}(X_i), & if \forall n, G_n^{alloc} \leqslant G_n \\ \infty, & else \end{cases} \tag{13}$$

In each iteration, we first generate children through crossover and mutation of genetic algorithm. Then we select particles with good fitness to form a new group and update the speeds and positions of particles according to (14) and (15). With the continuous iteration of the PSO-GA algorithm, we can finally find the approximate optimal scheduling strategy of the GPUs.

$$v_i = v_i + c_1 \cdot rand() \cdot (pbest_i - x_i) + c_2 \cdot rand() \cdot (gbest_i - x_i) \tag{14}$$

$$x_i = x_i + v_i \tag{15}$$

5 Evaluation

5.1 Experiment setup

In this section, we evaluate the performance of MagicBatch with real hardware testbed as emulation for the edge sever in the space or air. The arrival rate and deadline of tasks are randomly generated. We use VGGNet-16, GoogLeNet, ResNet-101, and DenseNet as models of inference tasks. The GPUs used in the experiment are Nvidia GTX 1080Ti, Nvidia RTX 2080Ti and Nvidia RTX 3090. The data set used in the experiment is CIFAR-10 and can be changed when necessary.

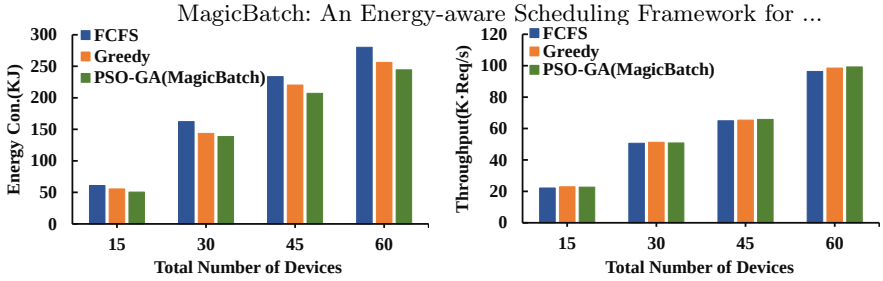
We evaluate the performance of MagicBatch from two aspects. First, we compare the proposed PSO-GA scheduling algorithm with the two baselines. (1) **FCFS**: The GPUs are numbered from small to large according to their advanced nature, and the idle GPUs with smaller numbers are allocated first. (2) **Greedy**: The tasks of each device are scheduled to be executed on the GPU with the lowest energy consumption among the currently idle GPUs.

Then, we compare MagicBatch with state-of-the-art baseline **TensorFlow Serving** (TF-Serving). We implement the scheduling policy in the official guide of TF-Serving [10], where batch size is a constant. We set it as 32, which is recommended in many papers [1, 2, 4]. When the number of tasks in the queue is equal to the target batch size, these tasks will be combined into a batch for execution. In addition, the scheduler will wait at most 30ms, otherwise, even if the number of tasks in the queue is insufficient, the existing tasks will be processed.

5.2 Effectiveness of MagicBatch

(1) Scheduling algorithm performance.

Figure 5(a) shows the energy consumption performance of PSO-GA scheduling algorithm finally used for MagicBatch and other baselines. With the increase of the number of tasks, the energy consumption is increasing for all algorithms. PSO-GA algorithm maintains the lowest energy consumption. This is because FCFS algorithm does not consider the difference in energy consumption of heterogeneous GPUs, resulting in high energy consumption. Greedy algorithm allocates the current task to the GPU that consumes the least energy during execution. However, this allocation method is not comprehensive. For a simple example, suppose there are two tasks task1 and task2. The optimal energy consumption for them to execute on GPU1 and GPU2 is 100J, 110J and 50J, 200J respectively. For the greedy algorithm, the first task 1 is placed on the GPU1 with the lowest energy consumption, and the second task 2 can only be placed on the GPU2, so the total energy consumption is 300J. Compared with these baselines, the PSO-GA algorithm can consider the problem from a global perspective. Although task 1 is not placed on GPU1 , task 2 saves more energy on GPU1. As the number of tasks increases, the situation becomes more complex, but such scheduling algorithm can minimize the overall energy consumption.



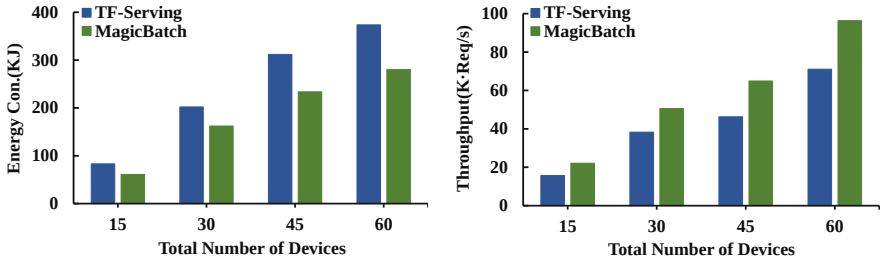
(a) Energy consumption (b) Throughput

Fig. 5: Performance comparison of different scheduling algorithms. PSO-GA is the scheduling algorithm used by our MagicBatch framework.

Figure 5(b) shows the throughput performance of scheduling algorithms. With the increase of access customers, the throughput of the server also increases. It should be noted that the optimization target of PSO-GA algorithm is total energy consumption. Therefore, the throughput may decrease slightly. But experimental results show that our scheduling algorithm can still achieve similar results with the other two scheduling algorithms, which shows that our scheduling algorithm is effective.

(2) Overall performance.

Figure 6(a) shows the performance comparison of MagicBatch and TF-Serving in terms of energy consumption. Similarly, with the increase of the number of tasks, the energy consumption of the GPU cluster is also increasing. The energy consumption is always lower when MagicBatch is adopted than TF-Serving is adopted (the optimization is more than 31.3 %), which reflects the advantages of the batching strategy. MagicBatch can select the optimal batch size according to the arrival rate and deadline of tasks in the scheduling process, and TF-Serving always maintains a fixed batch size. However, the batch size set in the initial stage is usually conservative, which leads to the fact that the parallel processing capacity of GPU is not fully utilized in the actual operation process, resulting in higher energy consumption.



(a) Energy consumption (b) Throughput

Fig. 6: Overall performance comparison.

Figure 6(b) shows the throughput performance comparison between MagicBatch and TF-Serving. With the increase of customers, the throughput of the server also increases. MagicBatch can always contribute a much higher throughput than TF-Serving (more than 41.1 %). This is also because MagicBatch makes

full use of the computing resources of the GPU by adaptively adjusting the batch size to obtain higher throughput.

6 Conclusion

More and more DNN applications are deployed on the heterogeneous edge servers in the space-air-ground networks. It is a significant challenge to energy-efficient schedule various DNN inference tasks on the GPU clusters. In this paper, we propose an energy-aware scheduling framework MagicBatch for DNN inference on heterogeneous edge servers. We model the energy consumption optimization problem of GPU clusters as an integer linear programming problem and propose a heuristic scheduling algorithm PSO-GA to allocate appropriate GPU computing resources for inference tasks. The experimental results show that MagicBatch can achieve more than 31.3% energy savings and 41.1% throughput improvement compared with existing methods.

References

1. Crankshaw, D., Wang, X., Zhou, G., Franklin, M.J., Gonzalez, J.E., Stoica, I.: Clipper: A low-latency online prediction serving system. In: 14th USENIX NSDI. pp. 613–627 (2017)
2. Cui, W., Wei, M., Chen, Q., Tang, X., Leng, J., Li, L., Guo, M.: Ebird: Elastic batch for improving responsiveness and throughput of deep learning services. In: 37th ICCD. pp. 497–505. IEEE (2019)
3. Dinh, H.T., Lee, C., Niyato, D., Wang, P.: A survey of mobile cloud computing: architecture, applications, and approaches. Wireless communications and mobile computing **13**(18), 1587–1611 (2013)
4. Fu, Z., Ren, J., Zhang, D., Zhou, Y., Zhang, Y.: Kalmia: A Heterogeneous QoS-aware Scheduling Framework for DNN Tasks on Edge Servers. In: IEEE INFOCOM 2022. pp. 780–789. IEEE (2022)
5. Jiang, J., Cui, B., Zhang, C., Yu, L.: Heterogeneity-aware distributed parameter servers. In: Proceedings of the 2017 ACM International Conference on Management of Data. pp. 463–478 (2017)
6. Jiang, Y., Zhu, Y., Lan, C., Yi, B., Cui, Y., Guo, C.: A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In: 14th USENIX OSDI. pp. 463–479 (2020)
7. Lin, B., Huang, Y., Zhang, J., Hu, J., Chen, X., Li, J.: Cost-driven off-loading for dnn-based applications over cloud, edge, and end devices. IEEE Transactions on Industrial Informatics **16**(8), 5456–5466 (2019)
8. Nabavinejad, S.M., Reda, S., Ebrahimi, M.: Coordinated batching and dvfs for dnn inference on gpu accelerators. IEEE Transactions on Parallel and Distributed Systems **33**(10), 2496–2508 (2022)
9. Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A., Zaharia, M.: Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In: 14th USENIX OSDI 20. pp. 481–498 (2020)
10. Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, F., Rajashekhar, V., Ramesh, S., Soyke, J.: Tensorflow-serving: Flexible, high-performance ml serving. arXiv preprint arXiv:1712.06139 (2017)

11. Park, J.H., Yun, G., Chang, M.Y., Nguyen, N.T., Lee, S., Choi, J., Noh, S.H., Choi, Y.r.: HetPipe: Enabling Large DNN Training on (Whimpy) Heterogeneous GPU Clusters through Integration of Pipelined Model Parallelism and Data Parallelism. In: USENIX ATC. pp. 307–321 (2020)
12. Yao, C., Liu, W., Tang, W., Hu, S.: Eais: Energy-aware adaptive scheduling for cnn inference on high-performance gpus. Future Generation Computer Systems **130**, 253–268 (2022)