Efficient GPU Resource Management under Latency and Power Constraints for Deep Learning Inference

Di Liu, Zimo Ma, Aolin Zhang and Kuangyu Zheng*
School of Electronic and Information Engineering
Beihang University, Beijing, China
Emails: {*diliu, mazimo, aolin2000, zhengky*}*@buaa.edu.cn*

***Abstract*—Recent rapid development in deep learning (DL) applications generates harsh requirements for DL inference services provided by GPU servers. On one hand, a high volume of different DL workloads always demands better processing throughput. On the other hand, GPU servers need to meet both the constraints of latency and power: each inference request must be responded in real-time with strict latency requirements; GPU servers need to be operated within a fixed power cap to prevent system failures from power overloading or overheating. Therefore, how to efficiently manage GPU resources to achieve better throughput under both latency and power constraints has become a key challenge.**

To address this issue, we first perform comprehensive measurements of inference tasks and have studied the impact of several critical knobs, including batch size, frequency, and GPU spatial sharing, on system performance in throughput, latency, and power. Then, we propose Morak, a multi-knob resource management framework for DL inference under the constraints of latency and power. A key mechanism of Morak is GPU resource partitioning with efficient space multiplexing for DL models. To further improve throughput, Morak efficiently explores the search space of GPU frequency and batch size under the constraints. Experiment results on a hardware testbed show that Morak can achieve as much as 67.7% throughput improvement compared with several state-of-the-art baselines under tight constraints of latency and power.

***Index Terms*—Deep learning, resource management, efficient computing, throughput, latency, power cap, spatial sharing.**

I. Introduction

The past few years witnessed the rapid development of many inspiring applications driven by machine learning (ML), especially deep learning (DL) technology, such as computer vision [1], natural language processing [2], recommendation systems [3], etc. As a trend, DL or in general ML-as-a-Service (MLaaS) has become an important service for servers [4]–[6]. The typical working scenario of DL includes two stages: training and inference. In the training phase, developers train DL models on massive datasets. Inference refers to the stage that data is input into the trained DL models to get the prediction results. GPUs have proven to be particularly suitable for these computation-intensive DL-based services due to their excellent parallel computing characterization [7]. To accelerate DL tasks, service providers are deploying large-scale GPU clusters to build fast parallel computing platforms.

Serving DL inference on GPU servers faces different challenges compared with training. It is well known that, due to the high price of hardware and high power consumption, both the Capital Expenditures (CapEx) and Operating Expenditures (OpEx) for high-performance GPU servers are very expensive. Therefore, in order to make the best value of the resources, service providers need to manage GPU resources reasonably to increase the throughput of executing inference tasks. Meanwhile, inference workloads are usually time-sensitive tasks for real-time applications directly and need to be completed within a strict latency limit to meet the Service-Level Objective (SLO) requirements. Violation of SLO can severely impact the user experience as well as the service provider's revenue. In addition, power capping is usually deployed on servers to avoid system failures caused by power capacity overloading or overheating. Strict server power cap need to be guaranteed during the execution of inference workloads. Therefore, based on the above challenges, it is a key issue to make full use of GPU resources efficiently to improve throughput under the dual constraints of latency and power.

Several methods have been proposed to improve GPU system throughput. Batching is a commonly used strategy [8], [9], which combines multiple inference requests into a batch for execution, the parallel computing capability of the GPUs can be better utilized. However, as the batch size increases, the latency of tasks also increases. Therefore, the value of the batch size is usually not large enough, which leads to underutilization of the GPUs and limits the further improvement of the throughput. Some other studies consider temporal sharing [10], [11]. Multiple DL models are mapped to the same GPU for execution, which avoids GPU idleness and improves the system throughput. However, due to the GPU time-slicing mechanism, only one model can be executed at a certain moment, which cannot solve the problem of insufficient utilization of GPU resources by a single model. The recent progress of the GPU architecture has been noticed by some studies [12], [13]. Starting from the Volta architecture, NVIDIA GPUs began to support the effective space division of GPU resources (called the NVIDIA MPS [14]). The mechanism supports the resources of GPUs to be divided to run different contexts at the same time. Therefore, it enables one GPU to serve multiple DL models concurrently, making it possible to further improve throughput. Unfortunately, the above studies only focus on improving throughput while meeting SLO requirements but ignore the power cap that servers need to guarantee. This may lead to

*Prof. Kuangyu Zheng is the corresponding author.

excessive server power consumption, which poses a large risk to the security of the system.

Meanwhile, Dynamic Voltage Frequency Scaling (DVFS) is a common method for managing CPU server power. Server power cap can be controlled by adjusting the frequency or voltage of key components such as processors and memory [15], [16]. For GPUs, vendors have developed user-friendly interfaces for frequency adjustment [17]. Some recent studies have begun to control the power of GPU servers through frequency adjustment [18], [19]. However, this kind of research only focuses on improving throughput under the power cap. Unfortunately, the frequency of the GPUs has a significant impact on the performance of running applications. Decreasing the frequency may lead to a violation of the SLO requirements of inference workloads, which in turn affects the revenue of the service providers.

Therefore, separately considering the constraints of latency or power is not sufficient for GPU servers. There is an increasing demand for an efficient GPU resource management framework that improves throughput under dual constraints. In this paper, we comprehensively measure inference workloads with various classical DL models. The results reveal several key knobs that affect the performance of inference workloads, including batch size, GPU frequency, and spatial sharing of GPU resources. Different from existing work, we also investigate the impact of spatial sharing on GPU power. At the same time, we find that the impact of these knobs on workloads performance is not isolated, that is, any one of the knobs has an impact on both latency and power. This means that latency and power are coupled with each other and need to be considered coordinately. Based on these observations, we propose **Morak**, a **m**ulti-kn**o**b **r**esource m**a**nagement framewor**k** for DL inference under the dual constraints of latency and power. A key mechanism of Morak is the efficient spatial sharing of multiple DL models. Besides, to further improve throughput, Morak scheduler searches efficiently in the variable space of GPU frequency and batch size. To our best knowledge, this is the first work to jointly manage both latency and power cap, to optimize the total throughput of the DL inference system. Specifically, this paper makes several key contributions:

- We perform comprehensive measurements on inference workloads with various classic DL models, and the results reveal the impact of batch size, GPU frequency, and GPU spatial sharing on inference workloads performance. To our best knowledge, this is the first work to investigate the impact of GPU spatial sharing on power.
- We propose an efficient GPU resource management framework called Morak to improve throughput while meeting latency and power constraints.
- We conduct an extensive evaluation of Morak on a hardware testbed. Experiment results show that Morak achieves as much as 67.7% throughput improvement under tighter constraints compared with several state-of-the-art baselines.

The rest of this paper is organized as follows. Section II surveys the related work. Section III illustrates the motivation of this article through detailed experiments. Section IV provides an overview of Morak. Section V introduces the design details. Section VI analyzes the evaluation results. Finally, Section VII summarizes this paper.

II. Related Work

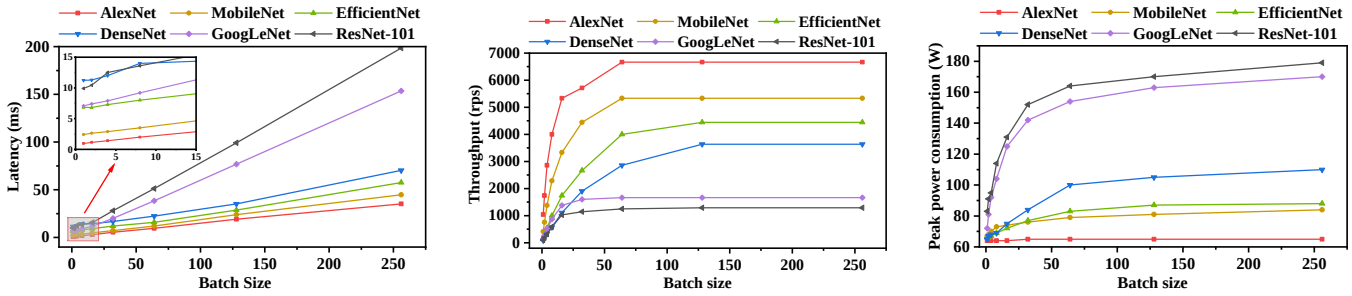
Currently, there are three categories of studies to improve the throughput of GPUs for inference workloads under SLO requirements, including batching, temporal sharing, and spatial sharing.

At the batching level, multiple inference requests are combined into a batch to execute, and the SLO requirements are guaranteed by adjusting the value of the batch size. Clipper [20] is the first framework using an add-increase-multiply-decrease (AIMD) scheme to dynamically adjust the batch size to maximize throughput under the constraint of the target SLO. Nanily [21] is a QoS-Aware scheduler with adaptive batching and autoscaling to achieve a balance between sub-second latency and resource efficiency in cloud servers. However, due to latency limitations, the value of the batch size is usually not large enough, which leads to underutilization of the GPUs.

At the temporal sharing level, different models or different layers of models share one GPU to avoid GPU idleness. Nexus [11] maps multiple DNN models to the same GPU for executing under the constraint of SLO. Prema [22] proposes a preemptible neural processing unit (NPU) and a predictive multi-task scheduler to meet the latency demands of high-priority inference while maintaining high throughput. Unfortunately, due to the GPU time-sharding mechanism, such methods cannot fundamentally solve the problem of insufficient utilization of GPU resources by a single model or single layer.

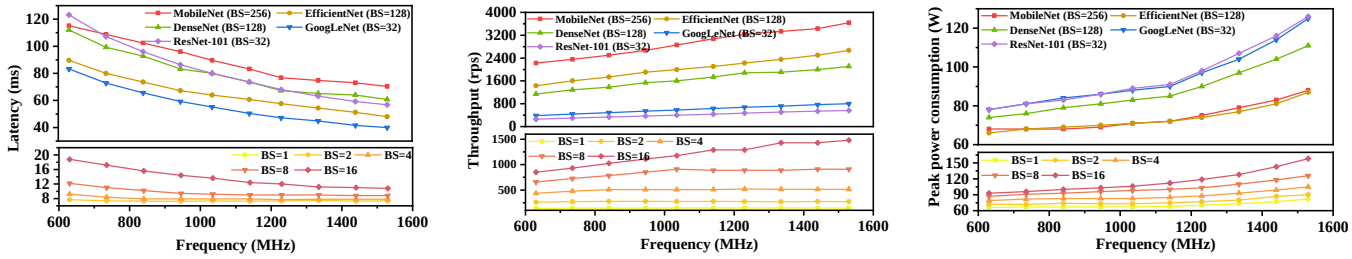
At the level of spatial sharing, the Ebird [23] inference engine supports the concurrent execution of different batches, which improves the utilization of GPU resources. Kalmia [24] divides different tasks into two CUDA contexts according to the requirements and uses a preemption-based scheduling method to reduce the deadline violation rate of DL inference workloads and achieve high system throughput. Gpulet [13] is the first work to comprehensively consider the batch size and spatial-temporal sharing to improve system throughput. Our work differs from the above studies, as besides latency, we also manage to constrain the power of the GPU servers, which is an ignored key factor.

Some previous studies have investigated server power capping methods. Lefurgy et al. [15] proposed a feedback controller to control server power consumption by adjusting CPU frequency. However, the processors it targets are CPUs instead of GPUs, and reducing the frequency leads to an increase in latency. BatchDVFS [18] improves throughput within the power cap by jointly considering the batch size and GPU frequency, but it cannot guarantee the SLO constraints. OptimML [25] is the first framework to jointly manage latency and power constraints, while its optimization goal is to improve the accuracy of inference workloads. Compared with



(a) Latency (b) Throughput (c) Peak power consumption

Fig. 1: Latency, throughput and peak power consumption under different batch sizes.



(a) Latency (b) Throughput (c) Peak power consumption

Fig. 2: Latency, throughput and peak power consumption under different GPU frequencies.

existing work, we design a GPU server resource management framework to improve system throughput under both latency and power constraints.

III. Motivation

In this section, we perform comprehensive measurements on inference workloads using various classical DL models. The results reveal several key factors that affect the performance of inference workloads. Note that, to our best knowledge, this paper is the first study to investigate the impact of spatial sharing on GPU power consumption. These important observations motivate us greatly to design our efficient resource management framework.

A. Batch Size

To evaluate the impact of batch size, we execute inference workloads of six DL models under varying batch sizes on a server configured with NVIDIA RTX 2080Ti GPUs. Figure 1 shows the comparison results of latency, throughput, and peak power under different batch sizes.

From Figure 1(a), it can be seen that the execution latency of DL inference workloads is approximately positively linearly correlated with batch size. As shown in Figure 1(b), increasing the batch size allows the GPUs to execute more inference tasks at the same time, contributing to a higher throughput. But as the batch size continues to increase until it exceeds the parallel processing capabilities of the GPUs, the throughput gradually levels off. Meanwhile, heterogeneous DL models have different demands on computing resources, and they reach peak throughput with different batch sizes (e.g., GoogLeNet: BS=32, MobileNet: BS=64). It can be seen from Figure 1(c) that the peak power consumption of the GPUs continues to increase.

Conclusion 1: Increasing the batch size can significantly improve the system throughput at first, but this comes with increased latency and power. After the throughput peaks, continuing to increase the batch size only brings the negative impact of increased latency and power. Therefore, an important conclusion we found is that the value of batch size should have an upper limit.

B. GPU Frequency

The second factor we measure is the frequency of the GPUs. While most previous power capping studies rely on DVFS on the CPUs, for DL inference workloads, the CPU utilization is low as the data is transferred to the GPU memory and the computation is mainly performed on the GPUs. Therefore, adjusting the CPU frequency will not have much impact on system power consumption. The GPU driver automatically adjusts the voltage based on the selected frequency, so we only need to change the frequency of the GPUs through the interface provided by NVIDIA. In order to ensure the validity of the conclusions, we perform measurements on different models and the same model (GoogLeNet) executed with different batch sizes under different GPU frequencies. The performance comparison is shown in Figure 2.

It can be seen that as the GPU frequency increases, the latency of executing DL inference tasks decreases, while the throughput and peak power consumption increase. This is because the increase in frequency leads to an improvement in GPU performance. It should be noted that as shown in the lower part of Figure 2, this phenomenon exists regardless of

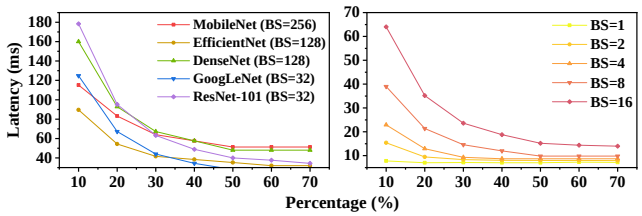


Fig. 3: Comparison of latency with respect to the percentage of GPU resources. The curves have obvious knee points.

whether batching is used or not. In addition, when the batch size is larger, the improvement effect is more significant.

Conclusion 2: We can increase the throughput of the system by increasing the GPU frequency, and this method also leads to lower latency. However, increasing the frequency will increase the peak power consumption of the GPUs, so it is necessary to adjust the GPU frequency appropriately to ensure that the power does not exceed the cap.

C. Spatial Sharing

We first analyze the case that the GPUs execute a single DL model at one time. As shown in Figure 1(b), the throughput plateaus only when the batch size is large. Actually, due to the latency limit and power cap, the value of the batch size can not be large enough, which means that a single DL model cannot fully utilize all the computing resources of the GPUs. In addition, we also measure memory usage during the execution of the DL inference workloads. Taking GoogLeNet as an example, when the batch size is 256, the memory usage is 3533MiB. However, the NVIDIA RTX 2080Ti GPUs we used have 11019MiB of memory. This shows that a single DL model cannot fully use the memory of the GPUs either. The same is true for bandwidth. The amount of data that needs to be transferred for a single DL model is far from enough to utilize the full bandwidth of the GPUs. Therefore, executing one model obviously causes insufficient utilization of GPU resources, and multi-model spatial sharing is necessary and could be more efficient.

Through the NVIDIA MPS mechanism, we change the percentage of allocated GPU resources and perform measurements on different models and the same model (GoogLeNet) executed with different batch sizes. Figure 3 shows the relationship between latency with respect to the proportion of allocated resources.

It can be seen from Figure 3 that the latency decreases with the increase of allocated resources. It should be noted that the slope of the latency curve changes significantly at a certain point (called knee point) as the percentage increases. The slope is larger before the knee point, and then rapidly decreases. The large slope means that the latency can be effectively reduced by allocating additional resources and the resource efficiency is higher at this time. This is due to the inherent communication overhead and non-parallelism of the different layers of the DL models. Within a single layer there are many parallel kernels that can be executed simultaneously in the GPUs, exhibiting a fraction of parallelism. However, DL inference is a process of forward propagation, and the output of the previous layer needs to be used as the input of the subsequent layer. As a result, the significant parallelism provided by high-performance GPUs cannot be fully exploited between different layers. In addition, for heterogeneous DL models, the position of the knee points is different. In our experiment, MobileNet is at 20%, GoogLeNet and ResNet-101 are at 30%. This is because different models exhibit different degrees of parallelism. It can be seen from the right part of Figure 3 that the larger the batch size, the greater the decline trend before the knee point, and the higher the effectiveness of resources.

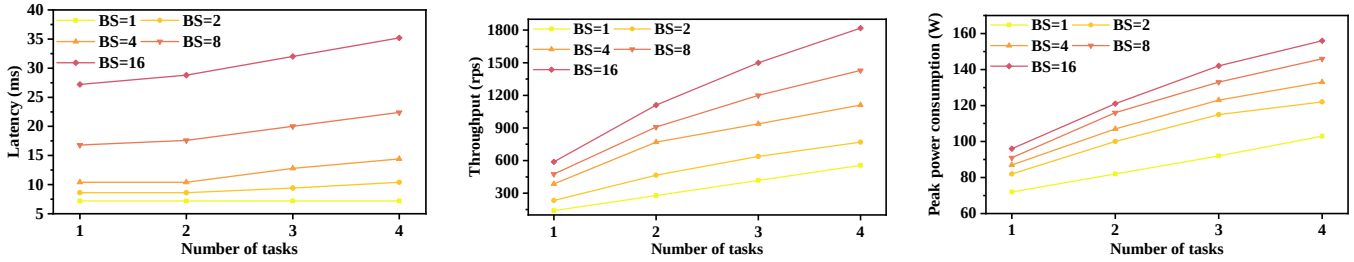
Conclusion 3: Providing a complete GPU for a single DL model is wasteful. Allocating an appropriate amount of resources based on the location of the knee point can achieve the best balance between performance and GPU resource consumption.

In addition to supporting the allocation of different proportions of GPU resources to processes, MPS also allows multiple processes to execute concurrently. Therefore, we can simultaneously execute the inference workloads of multiple DL models on the same GPU. In order to verify the impact of mutual interference, we measured the performance of GPUs executing multiple DL models concurrently. We allocate 25% of GPU resources to each process, increasing the number of concurrently running DL models from 1 to 4. We also compare the difference under different batch sizes. Figure 4 shows the latency, throughput, and peak power consumption with respect to the number of tasks.

It can be seen from Figure 4(a) that as the number of tasks increases, the latency of DL inference workloads increases. This is because MPS mechanism only logically isolates the GPUs, but does not achieve real physical isolation. Therefore, there is indeed mutual interference when multiple DL models are executed concurrently. It should be noted that the relationship between the latency and the number of tasks is approximately linear, but the slope increases when the number of tasks is equal to 2. It can be seen from Figure 4(b) that the total throughput of the GPUs also increases with the number of tasks. When the number of tasks is greater than 2, the slope of the curve decreases. From Figure 4(c) we can see that the increase in the number of tasks also leads to an increase in the peak power of the GPUs.

Take GoogLeNet as an example to illustrate the performance improvement that spatial sharing can bring. It can be seen from Figure 1(b) that GoogLeNet reaches a peak throughput of 1600 req/s when the batch size is equal to 64, the delay at this time is 38.4ms, and the peak power of GPUs is 152W. When we execute 4 models simultaneously in a spatial sharing manner and the batch size of each model is 16, we can obtain a total throughput of 1860 req/s, a latency of 35.2ms, and a GPU peak power of 156W. As a result, we achieve a 16.25% throughput and an 8.33% latency improvement at a 2.6% power consumption penalty.

Conclusion 4: Although mutual interference increases the latency, the spatial sharing of multiple models can effectively



(a) Latency (b) Throughput (c) Peak power consumption

Fig. 4: Performance comparison of GPU spatial sharing under multiple DL models (different workloads of GoogLeNet with varying batch sizes, e.g., BS=1, 2, ..., 16). Due to space limitation, only parts of the results are drawn here.

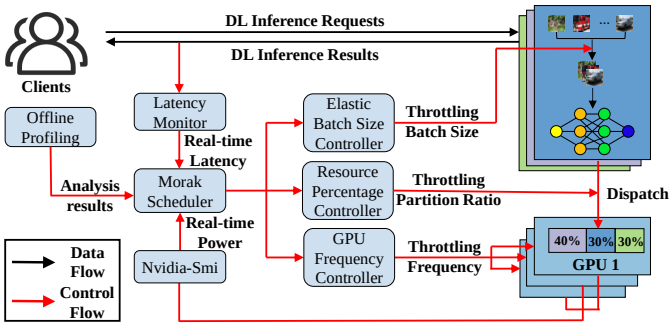


Fig. 5: Overview of Morak.

improve the total throughput of the system.

D. Summary

We summarize the key findings as follows: (1) Increasing the batch size within a certain range can improve throughput, but it will lead to higher latency and peak power consumption of the GPUs. (2) Increasing the frequency can improve the throughput and reduce the latency, but the peak power consumption of the GPUs will increase at the same time. (3) Spatial sharing of GPU resources across multiple models can effectively improve the overall throughput of the system. (4)The impact of GPU frequency and spatial sharing on the performance of DL inference workloads is more significant when the value of batch size is larger. The measurement results provide an crucial basis for the design of resource management framework.

IV. Morak System Overview

In this section, we provide an overview of the Morak system architecture. The goal of this study is a GPU server resource management framework for DL inference workloads. It needs to meet the latency limit of clients and the GPU power cap. We hope to make full use of GPU resources under the dual constraints and improve the total throughput of the system. The purpose of limiting the latency is to provide clients with SLO guarantees and improve service quality. The purpose of power capping is to prevent overloading or overheating of the power capacity and reduce the risk of system failure. In Section III, we analyze the impact of batch size, GPU frequency, and multi-model spatial sharing on the performance of inference workloads in detail. Obviously, we cannot simply consider the two constraints separately, because there is a coupled relationship between them. Therefore, to achieve efficient resource management, two constraints should be considered comprehensively.

Figure 5 shows the overall architecture of Morak. Clients send DL inference requests to the GPU server, and Morak is responsible for allocating GPU resources on the server to different clients' workloads. The core component of Morak is the scheduler. First, the scheduler allocates GPU resources for different DL models according to the results obtained in the offline analysis (where the knee point appears). Afterwards, the scheduler searches efficiently in the search space of GPU frequency and batch size.

In order to ensure that our framework can quickly respond to system changes (such as client SLO changes or server power cap changes, etc.), the monitor will keep running, and the scheduling algorithm will be called again when the system status does not meet the requirements. At the beginning of each scheduling cycle, the scheduler updates the resource allocation policy through the following steps: (1) Send the latency information of the DL inference workloads in the previous scheduling cycle to the scheduler through the latency monitor. At the same time, the peak power consumption of the GPUs in the last scheduling cycle is detected by the Nvidia-Smi tool, and the detection result is sent to the scheduler, either. (2) According to the difference between the detection results of the previous period and the current constraints, the resource allocation strategy is adaptively readjusted. (3) Set different knobs separately through the controller.

V. Design of Morak Scheduler

In this section, the design details of our Morak scheduler are presented. The goal is to maximize server throughput under the constraints of DL inference workloads latency and GPU power consumption. In addition, when the workloads or constraints change, the scheduler can update the scheduling strategy in the next scheduling cycle, so that the power and delay of the system converge within the constraints in a short time.

A. Problem Formulation

The problem that the scheduler needs to solve is as follows: given the latency limit L_u of the DL inference workloads and

the power cap P_c of the GPUs, Morak determines the resource sharing method and the frequency of the GPUs, as well as the batch size of each DL model, to maximize the overall throughput of the system. The problem can be expressed as follows:

$$Maximize: \frac{1}{T}\sum_{t=1}^{T} Th^t \tag{1}$$

$$s.t.: Th^t = \sum_{m=1}^{M} Th_m^t \tag{2}$$

$$L(b,f,p)\cdot L_{int}(n) \leqslant L_u^m \tag{3}$$

$$P_t(b,f,p)\cdot P_{int}(n) \leqslant P_c \tag{4}$$

Wherein, (1) represents our optimization goal. In order to ensure that the scheduling strategy is effective for a long enough time, the scheduling objective is to maximize the average throughput of T scheduling periods. Th^t represents the throughput of the system in the $t-th$ scheduling cycle. (2) indicates that the total throughput of a GPU in a scheduling cycle is equal to the sum of the throughputs of all DL models executed on it. (3) denotes latency constraints. L represents the latency function, which is related to the batch size b, the GPU frequency f and the allocated GPU computing resources p. We use the L_{int} function to represent the impact of mutual interference on the latency when executing multiple tasks through spatial sharing. (4) represents the power constraint. The P_t function represents the power of the GPUs at time t, and it is also related to the batch size, GPU frequency, and the spatial sharing strategy. P_{int} represent the impact of interference on the power.

B. Complexity analysis

The challenge in solving the optimization problem in V-A is that the performance of the system is affected by several inter-coupled knobs. For example, the latency of the workloads and the power of the GPUs are both affected by the batch size, and the benefits are negatively correlated, so there is a trade-off between them. The value of batch size is in turn influenced by the optimal amount of GPU resources required by the model. In addition, the allocation ratio affects the number of tasks executed in parallel on a single GPU. Therefore, the scheduling policy with the best performance lies somewhere in the multi-dimensional search space, which will bring huge search complexity.

Let N denote the number of GPUs on the server, P denote the number of possible GPU partition strategies, and F denote the number of selectable frequencies of each GPU. Therefore, the total number of possible partition and frequency settings for N GPUs is $(PF)^N$. Let M denote the number of DL models, each of which may be scheduled on at least one GPU partition. Let B denote the total number of possible batch size values for each DL model. Therefore, the complexity of the entire search space is at least $O((PF)^N B^M)$.

Since the search space is too large, an exhaustive search solution is impractical. At the same time, the coupling between variables makes it difficult to accurately establish the functional relationship. In addition, if the time complexity of the scheduling algorithm is too high, it may cause the inference workloads to timeout. To address this issue, we employ a heuristic approach to effectively reduce the search space.

C. Scheduling Algorithm

Algorithm 1 describes the scheduling process of the DL models. According to conclusion 3 in III-C, allocating more GPU resources to the DL models can reduce the latency and bring larger throughput. However, the latency curve has an obvious knee point, where the rewards of allocating more resources begin to decline. Therefore, in order to fully utilize resources, when adding a new DL model, we first allocate resources to the model in proportion to the knee point. In this way, a physical GPU is divided into multiple virtual GPUs. After that, we need to determine the frequency of the GPU and the batch size of the DL model executed on each virtual GPU.

Algorithm 1: Morak Scheduling Algorithm

Input: Power cap (P_c), latency limit (L_u), maximum frequency (F_{max})
Output: Optimal frequency (F) and batch size (B_n)

```
1  Set the GPU frequency and the batch size of each
     DNN model as the default value;
2  for each period do
3      Monitor the maximum power (P_max) of the GPUs;
4      if P_t^max ≤ α · P_c then
5          if β · L_u ≤ L_t ≤ L_u and F ≤ F_max then
6              Increase the frequency of the GPUs;
7          else if L_t<β · L_u then
8              Increase the Batch Size;
9      else if α · P_c ≤ P_t^max ≤ P_c then
10         Pass;
11     else
12         if β · L_u ≤ L_t ≤ L_u then
13             Decrease the Batch Size;
14         else if L_t<β · L_u then
15             Decrease the frequency of the GPUs;
16     The Frequency of GPUs can be modified only
         once in each period;
17 end
18 Reallocate GPU resources;
```

We first set the batch size to the default value. In order to ensure that the latency and GPU power do not exceed the limit, this value is usually small (e.g., equal to 1 and 600MHz). After running a cycle with this setup parameter, we compare the maximum power of the GPUs during that cycle with the power cap (Lines 1-2). Note that our Morak scheduler can satisfy both latency and power constraints, so the comparison of them does not need a fixed order.

In order to reduce the overhead of the system, we do not require the peak power of the system to be strictly equal to the power cap. This is because the peak power consumption of the

GPUs will fluctuate when executing DL inference workloads. Therefore, in order to avoid more overhead caused by frequent system adjustments, we leave intervals for the steady state ($\alpha \cdot P_c \leq P_t^{max} \leq P_c$).

If the peak power consumption is less than $\alpha \cdot P_c$, it indicates that there is still room for improving throughput. But at this time, we need to consider the latency. We first compare the latency of the workloads in the previous cycle with the latency limit. If it has reached a steady state ($\beta \cdot L_u \leq L_t \leq L_u$), We increase the throughput by increasing the GPU frequency because it does not negatively affect the latency (Lines 5-6). When the GPU frequency has reached the maximum value, the system has entered the optimal state to meet the constraints. If the latency is less than $\beta \cdot L_u$, it indicates that we can also increase the throughput by increasing the batch size at this time (lines 7-8).

If the peak power consumption is greater than the power cap, we continue with the latency comparison. If the latency is already approaching the latency bound, we reduce the power of the system by decreasing the batch size (Lines 12-13). If there is still a distance from the latency limit, we can reduce the frequency of the GPUs at the expense of the latency in exchange for a decrease in system power (lines 14-15).

Since we divide a physical GPU into multiple virtual GPUs through spatial sharing, changing the frequency of any one virtual GPU will cause the frequency of the remaining virtual GPUs to change. Therefore, in order to prevent instability caused by frequently changing the system frequency, the scheduler modifies the GPU frequency at most once in a scheduling cycle (line 16). In addition, if Morak's scheduling policy cannot satisfy the constraint of three consecutive cycles, the scheduler will adjust the spatial sharing policy of the DL models (Line 18).

The advantage of Morak is that in each scheduling cycle, the scheduler comprehensively considers the latency and power consumption constraints, and improves the throughput of the system as much as possible. The scheduling algorithm we adopt is very lightweight, which can ensure that the performance of inference workloads will not be affected by changing the scheduling strategy during operation. In order to ensure that the scheduling algorithm can adapt to system changes when the system latency and power consumption enter a stable state, the latency and power consumption monitor will continue to work. When the system status does not meet the constraints, the scheduling algorithm will be called again adaptively. Therefore, Morak also has good real-time performance.

VI. Hardware Evaluation

In this section, we comprehensively evaluate the performance of Morak. We first introduce the experimental setup. Then we compare Morak's throughput, GPU power consumption, and latency with other baselines. Finally, we discuss Morak's performance in terms of energy consumption.

A. Experiment setup

We perform all experiments on a server configured with four NVIDIA RTX 2080 Ti GPUs. Table I provides a detailed description of the server hardware and software. The RTX 2080 Ti GPU is based on the NVIDIA Turing architecture and supports the Nvidia MPS mechanism. Its peak power consumption is 250W. We use TensorFlow 1.15.0 as the DL framework.We use four classic image classification models as inference workloads. We use CIFAR-10 and ImageNet as a hybrid dataset. Note that Morak's performance does not depend on any specific feature, so Morak is applicable to other DL frameworks, DL models, or datasets.

TABLE I: Specifications of evaluation

Specifications Overview	
CPU	Intel i9-10900X @3.70GHz
GPU	4 × Nvidia RTX 2080 Ti
OS	Ubuntu 18.04.6 LTS
GPU Driver Version	470.161.03
CUDA Version	11.2
DL Framework	TensorFlow 1.15.0

We compare the performance of Morak with the following baselines. **Clipper-L [20]**: This method uses an adaptive batching strategy to dynamically find and adjust the maximum batch size, thereby satisfying the latency of inference workloads while improving the throughput of the system. For batch size selection, Clipper employs an Additive Increase Multiplicative Decrease (AIMD) scheme. When the latency is less than the limit, the batch size is increased by a fixed amount. A small multiplicative fallback is performed until the latency exceeds the limit, reducing the batch size by 10%. **Clipper-P**: This is a variant of Clipper that changes the batch size to meet the GPU's power cap. The original Clipper (Clipper-L) only considers the constraint of latency but ignores the constraint of power. For Clipper-P, we compare the peak power consumption of the GPUs with the power cap, and then use the AIMD scheme to dynamically adjust the batch size. **DFS**: This approach adjusts the frequency of the GPUs to meet the GPU's power cap without considering the latency constraints. Meanwhile, in this method, the value of the batch size is set to a fixed value. To ensure high throughput, this value is usually large. **BatchDVFS [18]**: This method considers both the batch size and GPU frequency to improve the throughput of the system under the power cap. However, this method does not consider the SLO requirement of DL inference workloads.

B. Performance of throughput

For this set of experiments, we compare the average throughput of Morak and the baselines executing different DL models. We set the power cap and latency limit to 150W and 100ms, respectively. Figure 6 shows the results.

It can be seen from the figure that our proposed Morak can achieve 130.9%, 110.3%, 67.7%, and 70.5% throughput

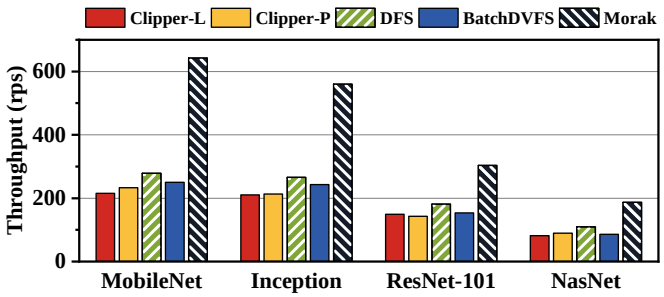


Fig. 6: Average throughput of Morak and baselines executing different DL models. Morak achieves the highest throughput.

improvement compared with other baselines. Considering the tighter latency and power constraints, this result is very exciting. An important reason for Morak's performance improvement is the adoption of GPU spatial sharing. By partitioning a physical GPU and executing different subtasks respectively, the GPUs can be utilized more efficiently. Although there may be mutual interference between models, the advantages brought by spatial sharing can offset the loss caused by interference and increase throughput. In addition, Morak's scheduling schemes are more fine-grained search results in three-dimensional space, so compared with BatchDVFS (two-dimensional) and other one-dimensional baselines, Morak can find better solutions.

C. Performance of constraints guarantee

In this set of experiments, we first fix the latency limit and adjust the power cap, and then fix the power cap and adjust the latency limit to evaluate the performance of Morak and other baselines in guaranteeing constraints. Figure 7 shows the evaluation results.

As can be seen from Figure 7(a), since Clipper-L only considers the latency limit, the peak power consumption does not change with the power cap. The value of the batch size is only relevant to latency. Due to the limited frequency range that the GPUs can choose, DFS is a coarse-grained control method. When the GPU frequency reaches the maximum value, it cannot be further adjusted. Therefore, as the power cap increases, the result first increases and then remains stable. Clipper-P and BatchDVFS do not consider latency constraints, so their batch size increases at the beginning, and power consumption increases. But as we discussed in Section III-A, there is no point in increasing the batch size too much and increasing the batch size after reaching the threshold will not increase the throughput. Therefore, when the batch size remains stable, the power consumption of Clipper-P also remains stable. BatchDVFS can adjust the frequency in addition to adjusting the batch size, so it flattens out later than Clipper-P. For Morak, since it adjusts the GPU frequency and performs space multiplexing on the GPUs, the peak power consumption continues to increase, making full use of the GPU resources. However, when the power cap is set too large, in order to ensure the latency limit, the power cannot continue to increase. Therefore, at some values Morak consumes slightly less power (e.g., 180W) compared to BatchDVFS. This does not mean

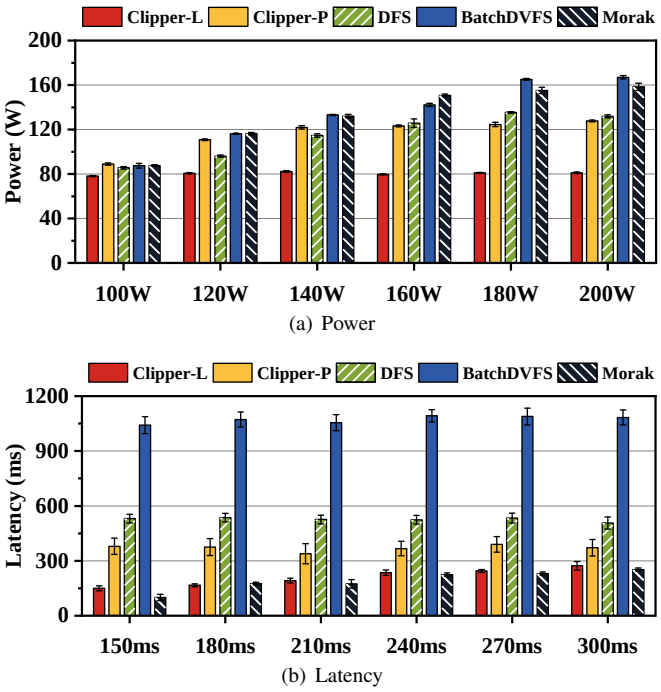


Fig. 7: Comparison at different power and latency setpoints. Morak can satisfy both power and latency constraints.

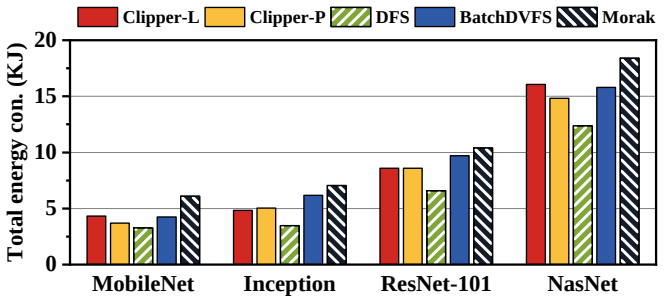
that Morak's performance is poor, because it additionally considers the latency limit. In addition, Morak can achieve higher throughput. The experiment results also show that too high a power cap is meaningless (e.g., 200W). Such high power consumption is not required when DL inference tasks is running.

It can be seen from Figure 7(b) that since Clipper-P, DVFS, and BatchDVFS only consider power constraints, the average job completion time (JCT) of inference tasks using these methods remains stable as the latency constraints change. This is because the scheduling results essentially depend on the fixed power limit we set. However, Clipper-L and Morak consider the latency. The JCT of Clipper-L continues to increase with the relaxation of latency limit, while Morak remains stable after increasing to a certain extent. This is because the power is approaching the cap at this time.

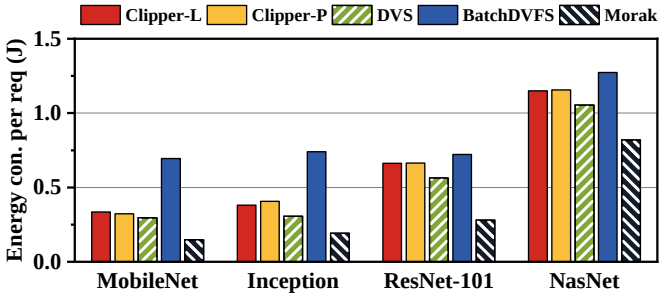
D. Performance of energy consumption

For this set of experiments, we compare the performance of Morak with baselines in terms of energy consumption. Although our system does not take energy consumption as a constraint, nor does it aim to optimize energy consumption, it is also meaningful to investigate energy consumption due to its strong correlation with power and task duration. We set the power cap and latency limit to 150W and 300ms, respectively. Figure 8 shows the total energy consumption of different methods running for the same duration and the energy consumption of each DL inference request.

It can be seen from Figure 8(a) that the total energy consumption of Morak after running for the same time is larger. But this result does not indicate that Morak's performance is



(a) Total Energy consumption



(b) Energy consumption per request

Fig. 8: Energy consumption comparision of Morak and other methods. Morak has the lowest energy consumption for processing a single inference request.

poor. It can be seen from figure 7(a) that Morak consumes more power during operation in most cases. Therefore, the energy consumption obtained by integrating Morak's power curve is greater than that of other methods. However, Morak's throughput is much higher than other methods. We can divide the total energy consumption by the number of inference requests executed by the system during this period to obtain the average energy consumption of each request, which is the result shown in Figure 8(b). From the figure, we can see that for various DL models, Morak requires the lowest energy consumption to execute a single inference request. Morak can achieve performance improvement of 49.8%, 37.4%, 50.2%, and 22.2% compared with the previous optimal baselines, respectively.

VII. Conclusion

Improving the throughput of DL inference workloads is a key issue of MLaaS. In order to guarantee the user's service quality, the latency SLO of the inference requests should be satisfied. In addition, power capping is set to prevent GPU server power overloading or overheating. In this paper, we propose Morak, a multi-knob efficient resource management framework that partitions the physical GPUs to execute multiple DL models concurrently in a spatial sharing manner. Besides, Morak selects the optimal GPU frequency and batch size to maximize the throughput of the system under the constraints of latency and power. Experiment results on a hardware testbed show that compared with the existing optimal baselines, Morak can achieve at least 67.7% throughput improvement under stricter constraints. Moreover, even though energy consumption is not the original optimization objectives, Morak manages to achieve 22.2% to 50.2% improvement of energy consumption per DL inference request, than the other state-of-the-art baselines.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016, pp. 779–788.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *ADKDD*, 2017, pp. 1–7.

[4] Alibaba machine learning platform for AI, 2023. [Online]. Available: https://www.alibabacloud.com/product/machine-learning

[5] Azure AI, 2023. [Online]. Available: https://azure.microsoft.com/en-us/overview/ai-platform/

[6] Amazon machine learning, 2023. [Online]. Available: https://docs.aws.amazon.com/machine-learning

[7] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[8] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "Tensorflow-serving: Flexible, high-performance ml serving," *arXiv preprint arXiv:1712.06139*, 2017.

[9] C. Zhang, M. Yu, W. Wang, and F. Yan, "MArk: Exploiting cloud services for Cost-Effective,SLO-Aware machine learning inference serving," in *ATC*, 2019, pp. 1049–1062.

[10] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for gpu with preemption support," in *RTSS*, 2018, pp. 119–130.

[11] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: A gpu cluster engine for accelerating dnn-based video analysis," in *SOSP*, 2019, pp. 322–337.

[12] A. Dhakal, S. G. Kulkarni, and K. Ramakrishnan, "Gslice: controlled spatial sharing of gpus for a scalable inference platform," in *SOCC*, 2020, pp. 492–506.

[13] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Serving heterogeneous machine learning models on Multi-GPU servers with Spatio-Temporal sharing," in *ATC*, 2022, pp. 199–216.

[14] NVIDIA Corporation, "Nvidia multi-process service," 2021. [Online]. Available: https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[15] C. Lefurgy, X. Wang, and M. Ware, "Server-level power control," in *ICAC*, 2007, pp. 4–4.

[16] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *HPCA*, 2008, pp. 101–110.

[17] NVIDIA Corporation, "Nvidia system management interface," 2023. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[18] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Coordinated batching and dvfs for dnn inference on gpu accelerators," *TPDS*, vol. 33, no. 10, pp. 2496–2508, 2022.

[19] C. Yao, W. Liu, W. Tang, and S. Hu, "Eais: Energy-aware adaptive scheduling for cnn inference on high-performance gpus," *Future Generation Computer Systems*, vol. 130, pp. 253–268, 2022.

[20] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A Low-Latency online prediction serving system," in *NSDI*, 2017, pp. 613–627.

[21] X. Tang, P. Wang, Q. Liu, W. Wang, and J. Han, "Nanily: A qos-aware scheduling for dnn inference workload in clouds," in *HPCC/SmartCity/DSS*, 2019, pp. 2395–2402.

[22] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *HPCA*, 2020, pp. 220–233.

[23] W. Cui, M. Wei, Q. Chen, X. Tang, J. Leng, L. Li, and M. Guo, "Ebird: Elastic batch for improving responsiveness and throughput of deep learning services," in *ICCD*, 2019, pp. 497–505.

[24] Z. Fu, J. Ren, D. Zhang, Y. Zhou, and Y. Zhang, "Kalmia: A Heterogeneous QoS-aware Scheduling Framework for DNN Tasks on Edge Servers," in *INFOCOM*, 2022, pp. 780–789.

[25] G. Chen and X. Wang, "Performance optimization of machine learning inference under latency and server power constraints," in *ICDCS*, 2022, pp. 325–335.