MISO: Exploiting Multi-Instance GPU Capability on Multi-Tenant GPU Clusters

MISO: Exploiting Multi-Instance GPU Capability on Multi-Tenant GPU Clusters

Baolin Li
Northeastern University

Tirthak Patel
Northeastern University

Siddharth Samsi
MIT Lincoln Laboratory

Vijay Gadepally
MIT Lincoln Laboratory

Devesh Tiwari
Northeastern University

ABSTRACT

GPU technology has been improving at an expedited pace in terms of size and performance, empowering HPC and AI/ML researchers to advance the scientific discovery process. However, this also leads to inefficient resource usage, as most GPU workloads, including complicated AI/ML models, are not able to utilize the GPU resources to their fullest extent – encouraging support for GPU multi-tenancy. We propose MISO, a technique to exploit the Multi-Instance GPU (MIG) capability on the latest NVIDIA datacenter GPUs (e.g., A100, H100) to dynamically partition GPU resources among co-located jobs. MISO's key insight is to use the lightweight, more flexible Multi-Process Service (MPS) capability to predict the best MIG partition allocation for different jobs, without incurring the overhead of implementing them during exploration. Due to its ability to utilize GPU resources more efficiently, MISO achieves 49% and 16% lower average job completion time than the unpartitioned and optimal static GPU partition schemes, respectively.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

KEYWORDS

Multi-Instance GPU, Resource Sharing, Multi-Tenancy.

ACM Reference Format:
Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2022. MISO: Exploiting Multi-Instance GPU Capability on Multi-Tenant GPU Clusters. In *SoCC '22: ACM Symposium on Cloud Computing (SoCC '22), November 7–11, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3542929.3563510

1 INTRODUCTION

Background and Motivation: Recent advancement in GPU technology has enabled HPC and AI researchers to leverage GPU computing capabilities for a wide variety of critical science missions, including training of compute-intensive neural network models [1–4]. While these advances have expedited the scientific discovery process, efficient resource utilization of the powerful GPUs remains a key bottleneck.

With innovative progress in computing technology, GPU vendors are making individual GPUs bigger and faster – where an individual GPU can now deliver more than 300 TeraFLOPS of performance and is on the path to becoming a supercomputer of the past by itself [5, 6]. This trend has served the AI/ML models well since the computing requirements of these models are increasing at a rapid pace [7–9]. Unfortunately, as our experimental characterization (Sec. 3) and previous works [10–14] have shown, even these models are not able to fully utilize the GPU computing resources, because various workloads have different resource bottlenecks and performance sensitivity to different resources. Therefore, the "one-size-fits-all" approach of making a single GPU more powerful is not optimal for all workloads and leads to inefficient resource utilization.

Recognizing and motivated by these challenges, GPU vendors have recently started offering native GPU resource partitioning capabilities to enable GPU workload co-location [15, 16]. These capabilities allow jobs to share the GPU resources concurrently and, thereby, reduce the cloud computing cost, reduce the long job queue wait time on HPC clusters, and potentially reduce the average job completion time (queue wait time + execution time). While promising, efficiently leveraging GPU partitioning is challenging because configuring a GPU to partition the resources optimally among co-located workloads is (1) cumbersome due to various practical partitioning constraints, (2) prohibitively time-consuming during the exploration process of finding a performance-efficient partition, and (3) incurs overhead (Sec. 2).

SoCC '22, November 7–11, 2022, San Francisco, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9414-7/22/11…$15.00
https://doi.org/10.1145/3542929.3563510

Therefore, the goal of this paper is to provide a novel method that automatically and quickly partitions GPU resources to achieve overall higher performance. Solutions in this space are expected to become increasingly critical as HPC centers are beginning to deploy modern GPUs with explicit resource partitioning abilities. For example, the NVIDIA A100 GPUs, which have MIG technology support, are a part of many cloud computing offerings, industrial research computing clusters, and academic HPC centers [17–20]. But currently, we do not have the tools to leverage MIG technology to effectively utilize MIG capabilities for faster execution and higher throughput, and thereby, reducing the cost of renting GPU resources or operating HPC clusters. Our proposed solution, MISO, is publicly available as an open-source package at *https://doi.org/10.5281/zenodo.7135988.*

Contributions. This paper makes the following contributions.

I. We present experimental evidence to demonstrate the opportunities and trade-offs in GPU workload co-location capabilities provided by modern GPUs and present a novel approach to exploit this trade-off. In particular, our experimental characterization of the Multi-Process Service (MPS) and Multi-Instance GPU (MIG) co-location capabilities shows that they offer different levels of partition granularity and performance isolation. Our experiments further reveal that determining the optimal GPU partition for the performance of co-located jobs is non-trivial and requires extensive exploration of interference-free MIG GPU configurations – incurring job disruption and overheads.

II. MISO is a novel mechanism that enables efficient co-location of GPU workloads using the recent advancement in the workload co-location capabilities on modern GPUs. To exploit the trade-off presented by different co-location capabilities, MISO approaches the problem of finding optimal GPU partitions for co-located workloads with a new perspective: *MISO proposes to use interference-prone co-location of jobs to estimate the near-optimal interference-free GPU partitions for co-located workloads.* This approach avoids the expensive exploration of different interference-free GPU partition configurations to determine the optimal partition.

MISO provides a learning-based method that can accurately estimate and predict an individual job's performance on GPU interference-free partitions (MIG configurations) from quicker, more flexible but interference-prone co-location capability (MPS configurations). MISO then leverages this information to dynamically determine the near-optimal GPU partition for co-located job-mix. MISO formulates this as a practical optimization problem, and schedules co-located jobs to improve key metrics of job completion time, makespan, and system throughput.

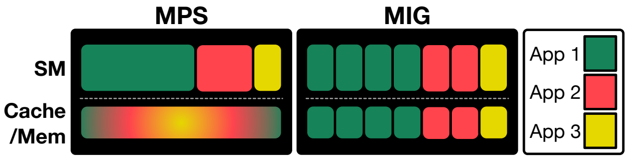

Figure 1: MPS sharing mode and MIG sharing mode.

III. Our extensive real-system and simulation-based evaluation confirm that MISO is effective at improving the key figures of merit (e.g., job completion time, makespan, and system throughput) under different scenarios. MISO's experimental evaluation is driven by representative production environments [10] and emerging workloads such as *BERT models for natural language processing, and Graph Neural Network (GNN) models for prediction of quantum chemistry molecular graphs* [21]. Our real-system evaluation demonstrates and explains why MISO outperforms existing techniques and its effectiveness is close to the practically-infeasible Oracle technique. MISO outperforms the unpartitioned GPU scheme by 49%, 15%, and 23% in terms of job completion time, makespan, and system throughput, respectively, and is within 10% of the Oracle technique for all three key metrics.

Next, we introduce the details of the state-of-the-practice GPU sharing technologies, in particular MPS and MIG, as well as the MIG capability on the latest NVIDIA A100 GPUs.

2 BACKGROUND

In this section, we introduce and compare different GPU resource partitioning paradigms on NVIDIA GPUs. *We acknowledge that we are using NVIDIA's GPU resource partitioning technology present in NVIDIA A100 GPUs as a vehicle to demonstrate the value of core ideas of MISO. We expect other GPU vendors to release similar capabilities in the near future as GPUs become increasingly powerful (Sec. 8). Along with these developments, MISO will continue to benefit current and future MIG-enabled cloud computing data centers and HPC centers [17–20].*

2.1 GPU Resource Sharing

Multiple applications can share one GPU using the time-sliced virtual GPU (vGPU) architecture. However, time-slicing does not address the challenge of running multiple applications that each cannot efficiently utilize a full GPU.

Multi-Process Service (MPS). MPS is a software-based space-sharing scheme that allows applications to run on the GPU simultaneously. It partitions the GPU compute units, streaming multiprocessors (SM), into multiple partitions (represented as % of total active threads), each partition is dedicated to a user application. MPS is the first-generation co-location support where a GPU can be configured to provide

Table 1: Complete list of MIG profile on an A100 GPU [16] (also refer to Appendix).

Slice	Compute	Memory	Cache	Max Count
7g.40gb	7 GPC	40 GB	Full	1
4g.20gb	4 GPC	20 GB	4/8	1
3g.20gb	3 GPC	20 GB	4/8	2
2g.10gb	2 GPC	10 GB	2/8	3
1g.5gb	1 GPC	5 GB	1/8	7

different levels of relative resource sharing among co-located workloads. *The resulting co-location is not interference-free* because, as shown in Fig. 1, only SM resources are dedicated to each application, and the cache and memory are shared among all.

Multi-Instance GPU (MIG). MIG is the latest hardware + software support for GPU resource sharing and partitioning supported on NVIDIA A100 Tensor Core GPUs [22]. MIG provides better isolation of different GPU resources among co-located workloads. Compared to MPS which only partitions the GPU SM, MIG also partitions the GPU memory, cache, and provides memory bandwidth isolation and error isolation between concurrent applications (Fig. 1). In other words, MIG allows the users to treat each MIG slice as a smaller A100 GPU with exclusive access, without the need to worry about performance interference with other user applications (i.e., *interference-free co-location*). However, this benefit comes with some *limitations*: (1) MIG only provides fixed partition sizes, the smallest partition unit on an A100 GPU with 40GB memory is `1g.5gb`, which provides 1/7 of SMs and 5GB GPU memory. MPS has a much finer granularity of SM partitions than MIG – the user can specify the amount of SM resource using any percentage integer. (2) When a new process arrives, re-configuring MIG to make space for the new application requires stopping all applications so that the MIG slices are idle. In MPS mode, a new application can be launched if enough memory exists.

2.2 NVIDIA A100 Tensor Core GPUs and MIG Capability

The A100 GPU's SM consists of 7 graphics processing clusters (GPC), in MIG mode, each slice (used interchangeably with MIG instance) includes at least one GPC and a corresponding amount of GPU memory. We list the full MIG slice profiles in Table 1. The max count means the maximum number of slices of the same type that can exist in the same GPU. The slice type notation shows the number of GPCs and the amount of GPU memory. Because the SM and memory are one-to-one mapped, we sometimes represent a slice with only the SM size (e.g., 4g) instead of the full notation. When we mention a larger/smaller slice, it means a slice with more/less number of GPCs, respectively.

Unlike the MPS approach, arbitrary MIG partitions are not supported due to hardware restrictions. A full A100 GPU is constrained to be partitioned only into certain combinations of MIG slices. For example, both (4g, 2g, 1g) and (2g, 2g, 3g) are valid combinations. However, due to hardware limitations, some combinations cannot exist even though the resources do not exceed the A100 cap, for example, `4g.20gb` and `3g.20gb` cannot co-exist in a single A100. In total, there are 18 MIG configurations on an A100 GPU (see Appendix). For a job mix (set of jobs to co-locate), the number of configurations is large because it includes not only the configuration of the MIG hardware, but also different assignments of jobs to the created MIG slices. Each such configuration is referred to as *partition configuration or MIG configuration.*

2.3 System Throughput, Job Completion Time, and Makespan

We briefly review the three widely used figures of merit relevant to quantifying the effectiveness of MISO and their definitions. When jobs are sharing a GPU in MIG mode, we use **system throughput**, or **STP** to measure the combined progress rate of all jobs [23, 24]. This metric essentially measures how much faster the jobs are progressing towards their completions (overall progress rate), compared to when these jobs are executed one by one in exclusive GPU without co-location. Formally, for m jobs J_1 to J_m, suppose job J_i's execution speed on an A100 GPU without co-location is p_i, and its current execution speed is q_i (on some MIG slice), then the system throughput is calculated as:

$$\text{System Throughput (STP)} = \sum_{i=1}^{m} \frac{q_i}{p_i} \tag{1}$$

This particular definition and similar variants have been widely used in the literature for denoting system throughput [23, 24].

Average job completion time (JCT) is the end-to-end service time of a job – the sum of the time spent waiting in the queue and job execution time. Average JCT is widely used to evaluate system software in the previous works [25–27], a shortened average JCT means users will experience better turnaround time and the system can support a larger user base.

Makespan is the time between the start of the first job to the completion of the last job in a job trace. These three metrics will be used to extensively evaluate system performance in Sec. 6.

3 MOTIVATION

In this section, first, we provide quantitative examples to demonstrate the potential benefits of partitioning GPU resources using the recently introduced Multi-Instance GPU

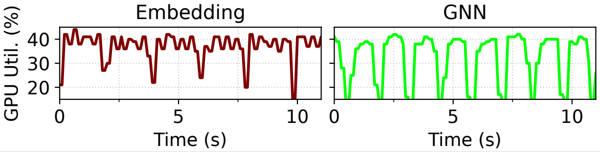

Figure 2: GPU resource utilization of example GPU applications.

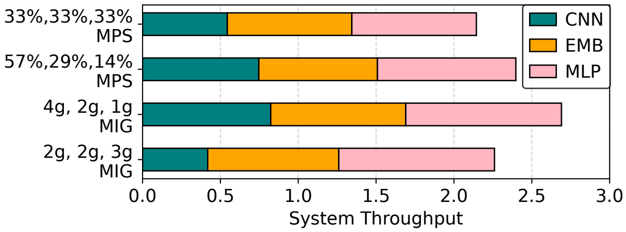

Figure 3: The system throughput of a workload mix with MPS sharing (top two bars) and MIG sharing (bottom two bars) on A100s.

(MIG) capability over the Multi-Process Service (MPS) method. Then, we discuss the challenges in achieving the full potential of GPU partitioning via MIG technology – MISO solves these challenges.

Takeaway 1. Many emerging compute-intensive workloads often cannot fully utilize compute resources in modern GPUs – motivating the opportunity for co-location.

Fig. 2 shows the GPU SM utilization for two representative workloads over their execution time (i.e., word embedding and graph neural network training). We note that the workloads often do not utilize the GPU resources at the maximum level. This is because modern GPUs are becoming increasingly powerful and provide higher computational power, but the workloads often have different bottlenecks (e.g., memory access latency, memory bandwidth) and hence, cannot leverage all the GPU resources to the fullest.

Takeaway 2. MIG's capability for workload co-location provides further opportunity for performance improvement beyond the MPS' method of co-location.

Fig. 3 shows the overall performance observed when three jobs (CNN, embedding, and multi-layer perceptron) are co-located as a job mix on an A100 NVIDIA GPU. The overall performance is indicated as system throughput (Eq. 1). First, we note that MPS-enabled co-location (first bar) allows multiple jobs to run together, and hence, achieve higher throughput than what would have been possible if co-located jobs were sequential (STP = 1). Second, we note that a MIG configuration (third bar) can provide higher system throughput than the MPS co-location. However, our MPS co-location (33%, 33%, 33%) shared the resources equally among co-located jobs, but the MIG partition (4g, 2g, 1g) divides the GPU computing resources in the ratio of 4:2:1. For a fairer comparison, we configure the MPS scheme to share the resources in the same proportion (second bar) and noticed that the MIG partition still yields higher performance. This is because the two workloads CNN and EMB both have seen improved performance even though the SM resources are the same as MPS, underlining the MIG's benefit of performance isolation and resource exclusivity among co-located jobs (illustrated in Fig. 1). We note that not all MIG configurations can outperform MPS configurations. A poorly-chosen MIG's system throughput (e.g., a workload needing the smallest memory capacity but assigned the largest MIG slice) will underperform MPS. For example, the (57%, 29%, 14%) MPS partition outperforms the (2g, 2g, 3g) MIG partition. However, MIG provides control knobs for partitioning different architectural resources, while MPS only controls the SMs and cannot control interference among co-located workloads for other resources (memory, bandwidth, etc.). Therefore, MIG is expected to outperform MPS in most cases. Our motivational example serves the simple purpose of demonstrating that better isolation achieved via MIG configurations can lead to higher performance.

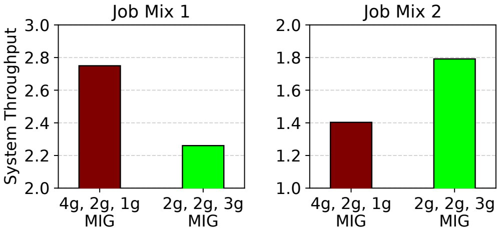

Figure 4: Sharing the GPU with different MIG partition patterns results in different performances (left). When the job mix changes, the optimal MIG partition may also change (right). Job mix 1 consists of (CNN, EMB, and MLP), while job mix 2 consists of (MLP, DeepSpeech, and GNN).

Takeaway 3. Optimal MIG partition configuration among workloads changes across different job mixes, but exploring for the optimal partition incurs prohibitive overhead – this is due to frequent GPU reconfiguration, high number of GPU resets, and I/O overhead due to repeated workload checkpoint-and-restart.

Fig. 4 shows the system throughput for two different job mixes running on two different MIG partition configurations. As one would expect, different partitions result in different performances for the same job mix. More interestingly, the performance ordering of the two MIG configurations is inverted for different job mixes. Therefore, when different mixes of workloads are co-located, the optimal resource configuration is likely to be different. It is critical to find the optimal GPU partition for a given workload mix.

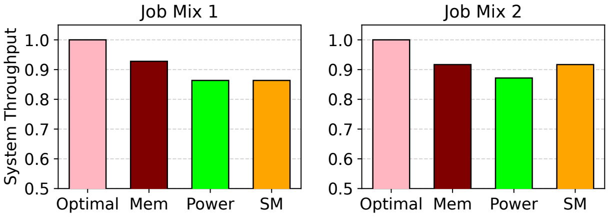

Figure 5: Applying heuristic-based approaches to perform MIG partition (using job's memory consumption, GPU power consumption, and SM utilization) does not yield the optimal MIG partitioning.

Unfortunately, finding the optimal GPU partition for a given workload mix is challenging because it requires experimentally evaluating the performance corresponding to different MIG partition configurations, which is time- and cost-prohibitive.

First, the number of possible MIG partition configurations is many and each configuration needs to be in effect for a certain duration for estimating its corresponding performance with high confidence. Second, each MIG configuration performance evaluation requires resetting the GPU, hence, disrupting the progress of all co-located jobs (it takes about 4 seconds for each GPU MIG reconfiguration). All jobs need to be restored back to their execution state when a new MIG configuration is put in effect. This requirement generates additional time and I/O overhead. The corresponding checkpoint overhead and the application restart time after MIG reconfiguration can be from seconds to minutes in practice. *In contrast, exploring different resource sharing levels of a job in MPS mode does not disrupt the execution of other jobs, all jobs in the GPU can execute concurrently in any MPS configuration.*

One can apply heuristic-based methods to avoid this evaluation process, but our experimental results show that such heuristic-based methods do not always guarantee to find an optimal partition. We design the heuristic to partition the GPU according to the job memory, GPU power consumption or SM utilization of each job when running exclusively on A100 GPUs. For each method (memory, power, or SM), we use the MIG partition whose number of GPCs has the highest Cosine similarity to the collected characteristic of the job mix. For example, if jobs in a job mix have memory sizes of 4000MB, 2500MB, and 1000MB, then we assign the partition (4g, 2g, 1g) to it because [4,2,1] has the highest Cosine similarity with [4000,2500,1000] than other partitions. Fig. 5 shows two examples where using the heuristic-based method to partition the GPU yields 8% to 14% lower system throughput than the optimal partition.

MISO takes a novel approach that combines the best of both worlds of MPS and MIG – MISO estimates the overall performance of different MIG configurations via quickly configurable MPS resource sharing levels instead of experimentally evaluating the MIG configurations exhaustively. This allows MISO to avoid the overhead of exploring different interference-free MIG resource partitions. Instead, MISO leverages the quick and more flexible, but interference-prone MPS GPU sharing mechanism to project the performance in the interference-free GPU partition situation (MIG). Ultimately, MISO uses this knowledge to determine the near-optimal MIG GPU resource partition to yield higher throughput and lower job completion time (more performance-efficient colocation reduces the job queue wait time).

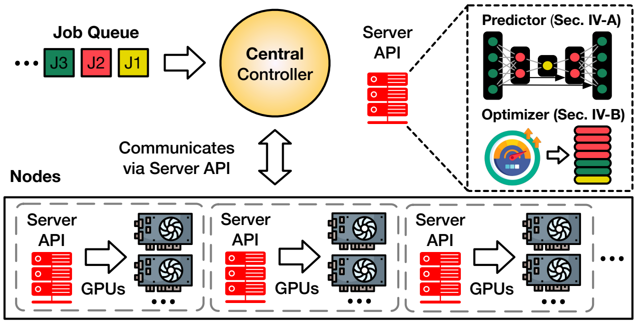

Figure 6: MISO design overview.

4 MISO: THE MIG SOLUTION

Fig. 6 shows an overview of MISO. MISO uses a central controller that monitors submissions from a job queue and communicates with server APIs distributed across nodes for status updates and scheduling decisions. Each server API corresponds to one MIG-enabled GPU. MISO uses a performance predictor (Sec. 4.1) to estimate every job's performance in a given job mix (set of co-located jobs) on different MIG slices using a learning-based predictor. It does so without running the job in the expensive, isolation-free MIG mode, instead, the co-located jobs are run only in the flexible, no-overhead interference-prone MPS mode only. Then, MISO uses these performance estimations to determine a MIG partition to maximize the overall performance, formulated as an optimization problem (Sec. 4.2).

4.1 MISO Performance Predictor

MPS-to-MIG Performance Estimation. The MISO predictor estimates a job's execution speed on different MIG slices (GPU partitions) relative to the maximum speed possible (i.e., when the job is run on the A100 full slice: `7g.40gb`). The key constraint is that MISO should not exhaustively run a given job on all possible GPU partitions (MIG slices) to generate the performance estimations for different MIG slices, because doing so would require frequently switching each job in the job-mix in and out of the GPU, incurring significant overhead and job idle time. To solve this challenge, MISO adopts a learning-based approach to build a model for predicting a job’s performance on all MIG slice types. The key idea is illustrated in Fig. 7.

At first, it might appear natural to train the learning-based model with different types of jobs in all possible MIG modes to make performance estimates on different MIG slices. However, recall that during the model-inference stage, we can not provide all the job performances on different MIG slices since that would require running each co-located job separately in interference-free MIG mode and incur GPU reset and checkpoint/restart overheads. Collecting these features is detrimental because jobs have to take turns to be profiled, while the other co-located jobs have to be stopped to make space for them. For example, assume there are five co-located jobs J1-J5 on an A100 GPU, to profile J1 on 7g, J2-J5 have to be paused. Similarly, for 4g/3g profiling, jobs have to take turns to be profiled, and the accumulated waiting time adds up.

Instead, MISO runs co-located jobs together in the MPS mode; then, it generates the model input features that are required to be used during the MISO’s model inference stage; then, the MPS-to-MIG performance estimation for each job in the job mix is combined to determine an effective partition configuration for the given job mix using a scheduler optimizer (Sec. 4.2). To confirm this experimentally, we measured the total profiling time for the number of co-located jobs using MIG-based profiling, which incurs up to 8× more overhead than MISO’s MPS mode profiling (in orders of minutes), to achieve similar scheduling quality as obtained by MISO. Also, as expected, MIG-based profiling gets worse as the number of jobs increases. In contrast, MISO retains near-constant cost due to concurrent execution of co-located jobs in MPS mode (shared contention-prone execution, but no GPU resets, no multiple rounds of evaluation, and fixed checkpoint-restart). In summary, MISO is more attractive than the MIG-based profiling because the MIG-based profiling requires frequent GPU resets and requires multiple rounds of evaluation since not all jobs can be profiled concurrently (and hence, multiple rounds of checkpoint-restart overhead and wait time).

Next, we discuss the key design trade-offs and lessons learned in designing the ML-based MISO performance predictor. MISO’s model should be able to estimate performance, not only the relative ranking, on all MIG slice types (interference-free execution with different resource configurations) with only interference-prone runs in the MPS mode (no performance isolation). We observed that translating to MIG performance from MPS runs requires us to be able to extract per-job interference-free high-level features from the interference-prone MPS profile. The interference-free high-level representation is needed because MIG provides hardware-level isolation between jobs. Using this as a motivation, MISO ’s design employs an autoencoder-based neural network because the center of the autoencoder represents the key abstract features. For example, we learned via experiments that collaborative filtering, widely used by previous resource schedulers [27, 28] is not suitable because they only produce the relative ranking, and other ML techniques such as linear regression, regression trees, and multi-layer perceptrons were not effective because they were unable to converge to an accurate state with limited input features.

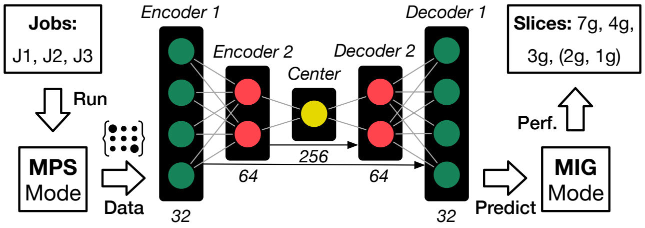

Figure 7: MISO predictor to translate MPS performance to MIG.

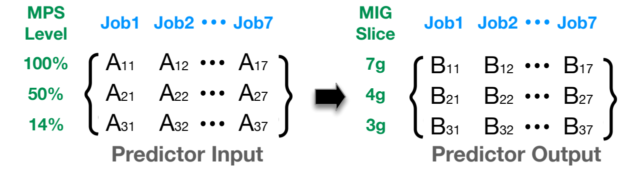

Figure 8: Input and output of the ML-based predictor.

Predictor Design. We construct a variant of the U-Net [29] convolutional autoencoder model. It is a lightweight model with fewer encoder/decoder blocks and fewer convolutional filters compared to typical models used in applications [30]. As shown in Fig. 7, the input is passed through two encoder blocks with 32 and 64 convolutional filters into its center with 256 filters, then through two decoder blocks into the translated MIG speedups. The convolutional filter size is 2×2 and the strides are (2,2) in horizontal and vertical directions.

Input and output. The inputs and outputs of the U-Net model are summarized in Fig. 8. The input to the model is a 3×7 matrix collected from MPS, corresponding to 3 MPS levels and 7 jobs running concurrently. The output from the model is also a 3×7 matrix, each column maps to a job, and the 3 rows represent performance on the 7g, 4g, and 3g MIG slices. For both the input and the output, each job (column) represents its execution speed at different MPS levels/MIG slices, normalized by the maximum speed in that column; all elements are within $(0, 1]$.

We set MPS active SM for all jobs at three different percentage levels: 100, 50, and 14. The intuition is to vary the amount of SM resources shared by jobs during MPS: at 100, all jobs share access to the full GPU; at 14, all 7 jobs have their own exclusive SM block partitioned by MPS; at 50, it is a middle ground between fully shared GPU SM and exclusive SM for every job. We set 7 as the number of columns (jobs) because the A100 GPU allows a maximum of 7 jobs in MIG mode. At each knob level, we profile the job execution speed for 10 seconds. An example of this speed is the number of mini-batches per second in the training of AI/ML applications.

Since the prediction model always takes 7 jobs (columns) as input to run in MPS, when there are less than 7 jobs, we pad the job mix with lightweight dummy workloads that we create until there are 7 total workloads. We use dummy workload padding instead of padding the input matrix with new columns of 0's because we find that large areas of zero padding greatly increase the training loss.

Memory considerations. Notice that the output only contains speedup information on 7g, 4g, and 3g slices. This is because some jobs cannot fit in the memory of 2g and 1g, while all MIG-compatible jobs will fit into 4g and 3g slices as both of them have the largest memory (20GB) of partitioned slices (7g is unpartitioned). For jobs that can fit into the 2g or 1g slices, we find that as long as we have the output on 7g, 4g, and 3g slices, the 2g and 1g output can be accurately predicted by a linear regression model from the other three slice types with an R^2 score of 0.96. Here $R^2 \in [0, 1]$ is coefficient of determination, where $R^2 = 1$ means the regression model can explain all the variations in the data perfectly.

Model training. To train the U-Net model, first, we need to collect training data for random job mixes running on both MPS and MIG modes. The data is collected by running randomly generated workload mixes (details in Sec. 5), whose job count ranges from 1 to 7. We generate 400 job mixes for each job count number, so in total, we have created 2800 job mixes for training. Each job mix is represented as a 3×7 MPS matrix input (with dummy filling) and a 3×7 MIG matrix as the target. We also perform data augmentation using the fact that the same set of jobs can be represented in different orders in the input/output matrix, but their MPS/MIG speedups will not change. Therefore, we create four extra different column permutations for each job mix – making the total data count 14,000. From the 14,000 data points, we randomly select 75% as training data and the rest as the validation set.

We train the model with mean absolute error (MAE) loss and Adam optimizer [31]. These hyperparameters along with others such as learning rate and activation function are selected using the ASHA hyperparameter tuning algorithm [32] on Ray Tune [33]. The validation loss converges quickly: we train the model for 50 epochs, and the validation loss (MAE) is 0.017, which is 1.7% over the MIG speedup target range. The training is also speedy: each epoch takes 3 seconds on one A100 GPU.

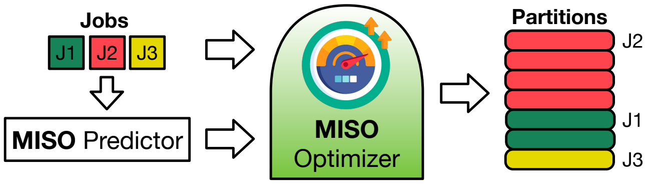

Figure 9: MISO optimizer to generate optimal MIG partition.

4.2 MISO Scheduling Optimizer

Now, we introduce the second component of MISO – which determines the GPU partitions (MIG slices) for a given job mix using the performance data collected purely in the MPS and the MISO performance predictor. When a new job is scheduled on a GPU, the GPU goes into MPS mode to profile the current job mix, and MISO solves an optimization problem to generate the new optimal partition configuration. First, we demonstrate how MISO formulates and solves the partition problem for a given job mix on a single GPU, and then, discuss the trade-offs in scaling it to the cluster setting.

GPU resource partitioning optimization. The goal is to use the MPS performance data collected for a short duration and performance predictor to determine the MIG partition for each job in a job mix on a single GPU. Suppose there are m jobs J_1 to J_m ($m \leq 7$ because each A100 can be partitioned into at most 7 slices), the partition configuration as the optimization variable can be represented as $\vec{x} = [x_1, x_2, ..., x_m]$ which has the same number of elements as number of jobs, x_i represents the MIG slice job J_i runs on. Since MIG partitions GPUs into pre-defined chunk sizes, $x_i \in \{1, 2, 3, 4, 7\}$, where each number corresponds to a unique MIG slice type (e.g., 1 means 1g, 7 means 7g). We do not make any assumptions about job execution time in the optimizer, in fact, when a job will run to completion is often difficult to predict [10, 26]. Hence, we judge the merit of a particular configuration x_i by the sum of each job's execution speed normalized to their maximum speed when running on an exclusive GPU. Our goal is to maximize the total system throughput for $\vec{x}$. Without the job speedup information on each slice type, this is a black-box optimization problem: we cannot know the performance of each configuration $\vec{x}$ unless we experimentally partition the GPU and assign the jobs on their corresponding slice to evaluate the overall performance. This is infeasible because every time we re-partition the MIG, checkpointing overhead occurs for the jobs. *This is why the performance estimator performs a key role for MISO – we do not need to keep reconfiguring and re-partitioning the jobs and the GPU during our optimization process.*

Fig. 9 shows an overview of the MISO optimizer that runs for each GPU. This optimizer is run to re-partition each MIG-enabled GPU whenever a new job starts on the GPU (during MPS), or whenever a job has finished execution to ensure that the GPU has no unused MIG slice at all times. With job speedup information on each slice type as input, the optimizer can immediately find the optimal partition without interrupting job execution during the process. The profiled information for each job i is represented as a function $f_i : x_i \longrightarrow k_i$, where $k_i \in (0, 1]$ is the job execution speed on MIG slice corresponding to x_i, normalized by the maximum speed on slice `7g.40gb`. For example, if J_1 runs 50% slower on `3g.20gb` compared to the full GPU, $f_1(3) = 0.5$. The problem is as follows:

$$\max_{\vec{x}} \sum_{i=1}^{m} f_i(x_i) \tag{2}$$

$$\text{s.t.} \quad \vec{x} \in P_{mig} \tag{3}$$

$$\|\vec{x}\|_0 = m \tag{4}$$

Here P_{mig} represents all the available partition configurations on an A100 GPU (from MIG documentation [16]). For example, $\vec{x} = [4, 2, 1]$ is a feasible MIG partition, so is $[4, 1, 2]$ because the physical partition is the same, the difference is that J_2 and J_3 are mapped to different slices. Thus, we use Eq. 3 to guarantee that the partition configuration is feasible. Eq. 4 means that the partition must have the same number of slices as the number of jobs – no slice bubbles or unscheduled jobs.

Algorithm 1 shows the pseudo-code that runs at each GPU upon job start and job completion. Because of its simplicity and lightweight, we do not observe any negative impact on the running jobs. The maximum optimizer runtime during our experiments is 0.5ms, negligible compared to jobs that run for orders of magnitude longer.

MISO for cluster setting. Optimizing job assignments on a MIG-enabled GPU cluster introduces a new dimension of complexity. Suppose there are n GPUs in the system, when scheduling jobs onto the n GPUs with MIG enabled, one needs to consider how to partition each GPU. Each A100 GPU can be partitioned in 18 different ways [16], thus the MIG configuration space is $O(18^n)$, which is exponential.

Therefore, globally configuring the MIG partitions across the whole cluster is a non-polynomial (NP) problem. Instead of tackling this NP problem, which could result in response time violation from the optimizer on large-scale systems, MISO simplifies it by locally solving a polynomial problem at every GPU. The reason it becomes polynomially solvable at each individual GPU is that the number of MIG configurations is capped at 18, and the number of jobs is capped at 7. One may hypothesize that this approach would miss out on the opportunity to migrate jobs among GPUs. However, based on our empirical experience, the performance gain from moving jobs between GPUs globally is not necessarily beneficial compared to the overhead. One overhead is from solving an NP-hard problem.

Algorithm 1: MISO's partition optimizer.

```
$best\_obj \leftarrow 0$ // Maximum objective so far
$best\_config \leftarrow None$ // Best partition so far
$P_{valid} \leftarrow$ list of $P_{mig}$ partitions whose length equals $m$
foreach $\vec{x}$ in $P_{valid}$ do
    $obj\_func \leftarrow \sum_{i=1}^{m} f_i(x_i)$
    if $obj\_func > best\_obj$ then
        $best\_obj \leftarrow obj\_func$
        $best\_config \leftarrow \vec{x}$
    end
end
return $best\_config$
```

The other major source of overhead is from extra checkpointing: when moving a job J_1 from GPU A to GPU B, GPU A needs to be re-partitioned so the jobs co-located with J_1 can access its resources; GPU B also needs to be re-partitioned to make space for J_1. Thus, all other jobs in GPU A and B will be checkpoint-restarted, causing systemic overhead. The performance gain from a better global configuration diminishes with the interruption of more jobs. In fact, our evaluation shows that even one-time checkpointing overhead can be significant enough.

4.3 Miscellaneous Design Considerations

Initial job placement and dynamic adaptivity. MISO monitors a first-come-first-serve (FCFS) queue and minimizes checkpointing overhead. It schedules a new job on the GPU that is hosting the least number of jobs. This policy aims to cause the least amount of disruption to all the jobs that are currently running in the cluster. When a new job is scheduled, the host GPU needs to go into the MPS mode for profiling, thus all jobs currently sharing the GPU in MIG mode will need to be checkpoint-restarted to run on MPS. Upon the profiling completion, the process repeats as the GPU switches from MPS back to MIG. Since the new job's execution characteristics are still unknown upon arrival, MISO attempts to minimize the negative impact on already running jobs.

We note that starting new jobs on the least crowded GPU helps with load balancing – all the GPUs in the cluster will host a similar number of jobs. It prevents the pathological case where multiple jobs are contesting for the resource of certain GPUs while other GPUs are underutilized. If MISO detects a significant change in execution speed for a running job (e.g., phase change), it will treat it as a new job and starts the MPS process for better repartition. MISO maintains configurable thresholds and historical data to ensure that re-invocations balance the trade-off between invocation cost and corresponding performance benefit from repartitioning.

Job out-of-memory. Different MIG slices provide different GPU memory sizes; some jobs may face out-of-memory errors when running on smaller slices. Users may specify the minimum GPU memory needed for each job. During the MPS stage, MISO also monitors the GPU memory usage for individual jobs using the `nvidia-smi` tool. The performance estimation from MPS then sets the corresponding speedup value to 0 before feeding the job information to MISO optimizer. For instance, if job J_1 cannot execute on `1g.5gb` slice, then the predictor sets $f_1(1) = 0$. The central controller maintains a "maximum spare slice" record for each GPU based on the memory constraints of its current jobs. It means that when re-partitioning the GPU, the maximum slice it can spare for a new job. When a job arrives in the queue with a memory limit, the controller will only consider GPUs whose "maximum spare slice" can satisfy the job memory constraint.

Quality-of-Service (QoS). The user may specify a minimum slice size that the job can execute on so that the MIG slice provides enough performance to meet the QoS constraints. MISO deals with this constraint similar to the job memory constraint, the central controller will only send it to GPUs that can squeeze out a new slice satisfying QoS.

Multi-instance jobs. In special cases, one job may spawn multiple instances of the same workload to run in parallel, such jobs naturally fit on multi-instance GPUs. MISO's performance predictor only runs for one instance of the job, then spawns all job instances on other GPUs using the profiled job information. The spawned instances do not need to be MPS profiled anymore – MISO directly starts the optimizer.

4.4 Implementation

MISO's implementation is built upon MPS and MIG APIs that we develop. Each GPU runs in MIG mode all the time because switching MIG mode on and off incurs extra overhead. When the GPU needs to run in MPS mode, it changes its partition to `7g.40gb` and runs MPS on top of the `7g.40gb` MIG slice. This capability to run MPS on top of MIG is supported by NVIDIA [16]. During MPS, the GPU keeps an MPS control daemon in the background. To connect a job to the MPS daemon, we pass the `CUDA_MPS_PIPE_DIRECTORY` variable to the job, which points to the same variable value specified by the daemon. To set the MPS level, we pass another variable `CUDA_MPS_ACTIVE_THREAD_PERCENTAGE` with the MPS level percentage as a value to the job. The MIG API is more involved and richer as configuring the GPU from one MIG partition to another involves a series of commands to destroy compute and GPU instances, then create new GPU and compute instances. To ensure a job starts on the correct MIG slice, we also need to retrieve the UUID using `nvidia-smi` commands, as this UUID varies across different MIG devices and different GPUs. We use an automated script to collect the MIG device UUID for each partition and stored as lookup tables (only needed once). To assign a job to a particular MIG slice, we pass the `CUDA_VISIBLE_DEVICES` variable with the corresponding UUID to the job. We have integrated these commands into Python function calls using the `subprocess` module.

For each job submitted to the system, because all the MIG slice assignments and MPS control tasks are implemented by passing environment variables to the job, the user does not need to make additional code changes. MISO's server API (Fig. 6) hosts a trained U-Net model in TensorFlow, and a partition optimizer utility. The GPU nodes do not communicate with each other, but they continue to update their status (i.e., job completion, current partition, MPS start/finish) to the central controller via TCP, so that the controller can decide the appropriate location for the next job.

5 METHODOLOGY

Evaluation Setup. We conduct real-system evaluations of MISO on an experimental testbed with four nodes, each node is equipped with 2 AMD EPYC 7542 CPUs and 2 NVIDIA A100-PCIe-40GB GPUs – thus, 8 A100 GPUs in total. Note that one A100 GPU is reported to be comparable to 3 NVIDIA V100 GPUs or 10 NVIDIA P100 GPUs for datacenter applications [34]. With 56 MIG slices in total, our testbed can serve up to 56 jobs at any given time.

We also perform an extensive simulation-based evaluation to test MISO's effectiveness on a 40-GPU A100 cluster. Simulation results are particularly of high significance since they show that MISO's benefits are not limited to small-scale systems. In Sec. 6, we conduct the simulation for 1000 different trials with a unique seed each time and report the results with violin plots and error bars.

Workloads. MISO's evaluation is driven by a job trace that emulates production behavior, in particular, our evaluation job trace is modeled after the most recently released and publicly available production GPU job trace for reproducibility and enhancement (Helios Trace [10]). For testbed experiments, we generate a 100-job mix that mimics the job execution time from the original trace when running on unpartitioned A100 GPUs. To accommodate the GPU hour time constraint, we limit the maximum job duration to be within 2 hours, which is approximately the 90^{th} percentile execution time of the Helios Trace. Note that a 2-hour job on A100 GPUs could execute for 5 hours on smaller MIG

Table 2: Workloads used to evaluate MISO.

Model	Batch Sizes	Application
ResNet50 [36]	64, 128, 256, 512	Image classification with residual learning
MobileNet [37]	64, 128, 256, 512	Image classification on lightweight model
BERT [38]	2, 4, 6, 8	Sentiment analysis of the IMDB movie reviews
Transformer [39]	16, 32, 64, 128	Time series prediction of engine noise measurement
DeepSpeech [40]	2, 4, 8, 16	Automatic speech recognition of the LJSpeech dataset
Embedding [41]	64, 128, 256, 512	Word embedding model for message topic classification
Graph NN [21]	64, 128, 256, 512	Property prediction of quantum chemistry molecular graphs
CycleGAN [42]	1, 2, 3, 4	Learning of mapping for image-to-image translation

slices, thus this limit helps us guarantee the completion of one set of experiments within a day. The job arrival follows a Poisson distribution with a λ of 60 seconds. Poisson distribution is widely used to model job arrival in multiple previous works [26, 27, 35].

For simulator evaluations, we use the same trace to generate a 1000-job mix for each trial and use a λ of 10 seconds for the Poisson distribution. The λ parameter is also swept over a range of values to model different job arrival intensities in different situations. When repeating the simulation for 1000 trials, the job generation is fully randomized with different job mixes, arrival orders, and execution times.

We use various types of deep learning (DL) workloads because the recent advancement in DL algorithms has made them popular in scientific research and production datacenters [43–45]. We uniformly sample the DL model and training batch size from Table 2. These workloads come from Hugging Face [46] and the `keras.io` repository [47]. The application domains include computer vision, language modeling, speech recognition, and scientific computing, and have distinctive DL operators including CNN, RNN, and embedding tables. *Disjoint sets of jobs mixes were used for training and testing.*

Competing Techniques. As MISO is the first work to exploit MIG features for datacenter operation, we devise two intuitive competing techniques and one oracle scheme as below.

NoPart. This scheme does not perform MIG partition on A100 GPUs, reflecting the default GPU usage scenario in datacenters. It is simple to operate: when a system upgrades its GPU hardware to A100s, the system operator can manage them just like the previous GPU generation.

OptSta. This scheme partitions all A100 GPUs into a fixed configuration that does not change over time. It is a straightforward way to manage MIG-enabled GPUs as a recent work Abacus [48] has deployed static partitions of (4g, 2g, 1g) on their A100 GPUs. However, the best MIG configuration changes when running different job traces. To make sure we always use the optimal static partition when comparing against MISO, we exhaustively evaluate all possible MIG configurations offline and choose *the best static partition*. Thus, the scheme is called optimal static (OptSta).

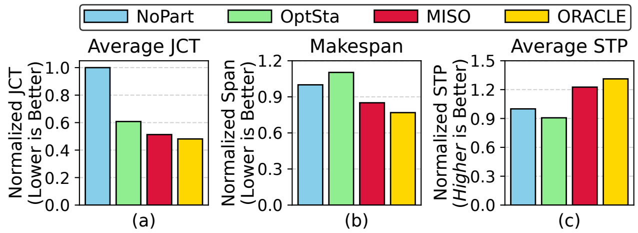

Figure 10: Performance comparison with competing techniques. All values are normalized to NoPart.

Oracle. This is similar to MISO except that it uses oracle information about job profiles of MIG slice speedups, which are collected offline before execution. Hence, it does suffer from profiling overhead and prediction inaccuracies.

Our evaluation results for OptSta and Oracle schemes do not include any profiling/switching overhead (ideal results), but our MISO results include its overhead for conservative performance improvement reporting. OptSta is the "best static MIG configuration" which works the best on average across all the job mixes (a single configuration). Oracle finds the best dynamic MIG configuration - different for different mixes. Therefore, MISO can be expected to outperform OptSta sometimes, but not Oracle.

Metrics. As discussed and defined earlier in Sec. 2.3, we use three widely-used figures of merit: **average job completion time (JCT)**, **makespan** and the **system throughput (STP)**.

6 EVALUATION

6.1 Real System Evaluation and Analysis

First, we present results and analysis on a real cluster to demonstrate the effectiveness of MISO and derive insights.

Job completion time, makespan, and system throughput. Fig. 10 shows the average job completion time for different competing strategies, normalized to unpartitioned (NoPart) strategy. Recall that the unpartitioned strategy does not create MIG instances to co-locate jobs. We make several observations.

First, the optimal static partitioning (OptSta) outperforms unpartitioned scheme by 39% (the absolute average JCT for NoPart is 40 minutes) (Fig. 10(a)). Recall that OptSta determines the optimal MIG instance partitioning via an exhaustive offline search process. The same partition is assigned

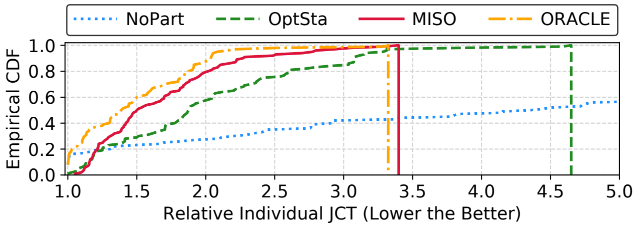

Figure 11: CDF of relative JCT of individual jobs. Each job's JCT is normalized to JCT when running on exclusive A100 GPU without queuing delay. The vertical line represents the maximum.

to all the GPUs in the cluster, and this partitioning leads to better JCT for all jobs on average. Second, we observe that MISO significantly outperforms even this offline strategy by 16%, even though MISO does not utilize any oracle information or require offline processing for decision-making. This is because MISO determines the GPU resource partition dynamically and specifically targets a given job mix. It also adjusts its partition as the job mix changes (e.g., arrival of new jobs, completion of existing jobs). Finally, our results show that MISO achieves similar performance to the Oracle strategy, which is dynamic and utilizes futuristic information.

Fig. 10(b) and (c) reflect similar trends for the other two figures of merit: makespan and system throughput. MISO shortens the makespan by 23% over optimal static partitioning and is within 10% of the ORACLE strategy. Similarly, MISO increases the system throughput by 35% over OPTSTA and stays within 7% of the ORACLE strategy. OPTSTA outperforms the no-partition strategy in terms of JCT but performs worse than the no-partitioning strategy in terms of system throughput and makespan. This is because a few long-running jobs cannot access extra GPU resources when they are uncontested in OPTSTA, so they become straggler jobs with a long makespan. Nevertheless, MISO outperforms both NOPART and OPTSTA strategies in all aspects and is similar to the Oracle strategy.

While Fig. 10 confirms that MISO outperforms competing techniques, it only provides the average improvement across jobs. Next, we provide deeper quantitative evidence to demonstrate MISO's effectiveness. Fig. 11 shows the relative JCT for all jobs in the trace compared to their isolated, interference-free execution on the full GPU – represented as the cumulative distribution function (CDF) of relative JCT for all competing techniques. This result confirms that overall average improvement in JCT (Fig. 10(a)) is not a result of MISO's aggressive attention to certain jobs. In fact, Fig. 11 shows that MISO consistently provides improvements for all jobs compared to all schemes. Similar to the Oracle strategy, 50% of MISO's jobs experience within 1.5× of the ideal JCT they can possibly have without sharing and queuing, while for NOPART and OPTSTA, this portion is less than 30%.

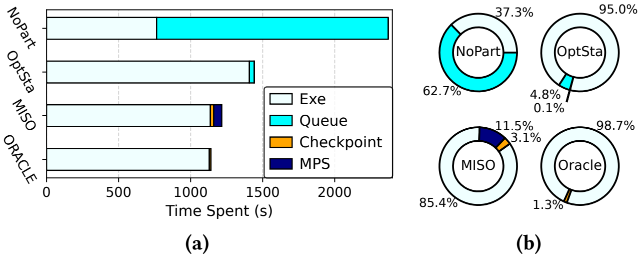

Figure 12: Breakdown of stages during the entire job life cycle. (a) shows the absolute time and (b) shows the percentage time.

Next, we dig deeper to understand the reason behind MISO's strength.

Why does MISO perform effectively? We use two different experimental pieces of evidence to demonstrate the key sources of MISO's effectiveness. First, Fig. 12 shows the breakdown of average job completion time spent in various stages for different competing schemes. As expected, jobs in the no-partition scheme spend over 60% of their total time in the queue because of the unpartitioned GPU resources. While the jobs benefit from running colocation-free on these GPUs, ultimately, their queue wait time negatively affects the overall JCT. OPTSTA reduces the queue wait time by allowing effective co-locations, but because of its static nature, OPTSTA still remains sub-optimal and the jobs spend about 5% of the time in the queue due to limited processing capability. Note that OPTSTA also migrates jobs from small slices to larger slices upon availability, but the checkpointing overhead is negligible (0.1%). In contrast, MISO and ORACLE completely eliminates queue wait time, providing evidence for their outstanding job processing power, and capability to support larger user bases. This is realized by incorporating dynamic partitioning across different GPUs depending upon the co-located job mix, instead of one single static partition across all GPUs in the cluster. However, MISO incurs extra checkpointing overhead because it requires jobs to run in MPS mode to estimate the optimal GPU partition. The job is still progressing towards completion during MPS mode, thus MISO is able to keep up with the pace of ORACLE even though MPS accounts for 12% of the time.

This result also highlights the importance of MISO's approach of using MPS mode to reduce MIG configuration explorations – this reduces the needed checkpoint to only 3% in Fig. 12(b). If we do not start with MPS mode but choose to exhaustively profile the job speedups in MIG, this fraction grows to more than 20% while jobs also experience significant idle periods during this process. This means that frequent checkpoints needed to explore different MIG partitions to determine a near-optimal partition is time prohibitive – hence, highlighting the importance of MPS to MIG translation.

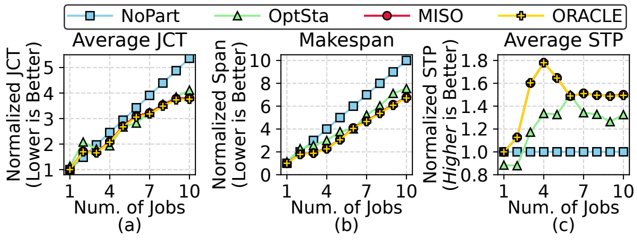

Figure 13: Scheduling more vs. less jobs on a GPU.

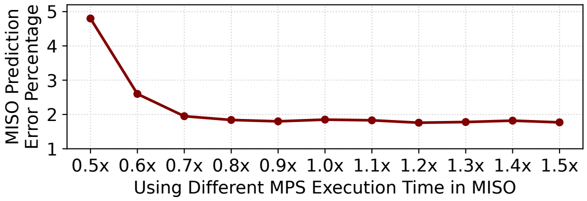

Figure 14: MISO's prediction error when changing the MPS profiling time. 1× represents MISO's current MPS time at 10 seconds per MPS level (Sec. 4.1)

Next, we conduct a single-GPU experiment to show MISO's effectiveness as we increase the number of jobs scheduled for the GPU: we conduct 10 trials with increasing job number from 1 to 10, and each job lasts for 10 minutes on an exclusive A100 GPU. We show the results in Fig. 13, where all metrics are normalized to the 1-job NoPart trial. Because NoPart processes the jobs one by one, its average JCT and makespan follow a linear trend as the number of jobs increases, and its system throughput remains 1 due to no GPU sharing. First, we observe that the difference between MISO and NoPart broadens as the number of jobs increases, meaning MISO is more capable of processing heavier workloads. Second, the OptSta scheme could even occasionally outperform MISO in JCT, this shows that OptSta is a highly competitive scheme for some job mixes. However, in a system with a large number of GPUs and jobs, it is unlikely that every GPU can receive a job mix that matches well with the same static partition – MISO resolves this issue with its job-mix-specific partition optimization at each GPU. Finally, almost all MISO and Oracle data points overlap in Fig. 13, meaning MISO has found the oracle partition for most job mixes.

Benefit from longer MPS execution time yields diminishing returns, and MISO provides a significant advantage over the MPS-only approach. Recall that MISO leverages brief MPS-mode execution to estimate the optimal MIG partition. Fig. 14 shows the effects of increasing and decreasing MISO's MPS profiling time. When the MPS time is cut to half (0.5×), the prediction error becomes much higher. But further increasing MPS profiling time only yields diminishing returns in prediction accuracy. At 1.5× MPS profiling time, we have even observed a 4% performance degradation in JCT, this is because the system does not have accuracy benefit from the longer MPS time, but experiences longer inefficient execution in MPS compared to running on optimal MIG partitions. Later in Sec. 6.2 we will also show that MISO is tolerant to a larger prediction error.

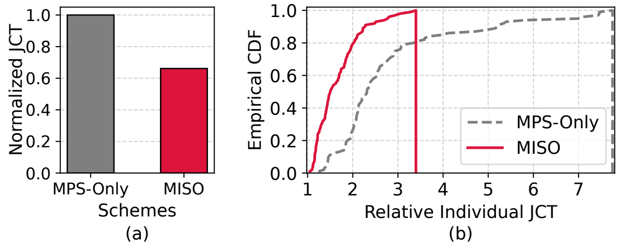

Figure 15: Comparing MISO against an MPS-only baseline. (a) shows the average JCT normalized to MPS-only. (b) shows the CDF of relative JCT of individual jobs compared to exclusive execution on full GPU.

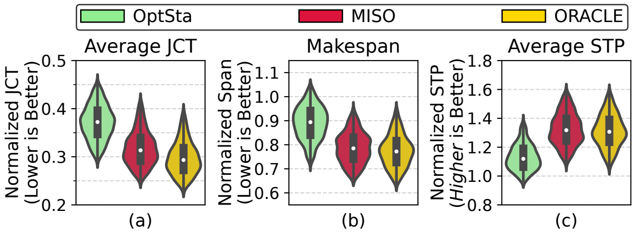

Figure 16: Violin plot of the results during the 1000 repetitions.

MISO achieves significant performance gains from the unpartitioned GPU baseline. In Fig. 15, we compare MISO against an MPS-only baseline partition to show that MISO's benefits stem from intelligently partitioning the GPU resources using the MIG technology. The MPS-only scheme partitions each GPU's SM into three equally sized portions (limiting to three because more partitions lead to worse performance and out-of-memory error), and co-locates the jobs on these MPS partitions. Fig. 15 (a) shows that MISO improves the average JCT by 35% compared to the MPS-only baseline. Fig. 15 (b) shows the relative JCT for individual jobs (same as Fig. 11) have shorter JCT when running on MISO – 80% of jobs have less than 2× JCT degradation compared to exclusive A100 execution on MISO while the corresponding portion is 30% on MPS-only.

6.2 Simulation Evaluation and Analysis

Our simulation evaluation tests MISO's effectiveness under different scenarios and at a larger scale (40 GPUs, 1000 jobs) that is cost-prohibitive on real systems.

Job completion time, makespan, and system throughput. Our evaluation particularly focused on validating the consistent performance improvements by forcing each simulation run to start with different initial conditions (different

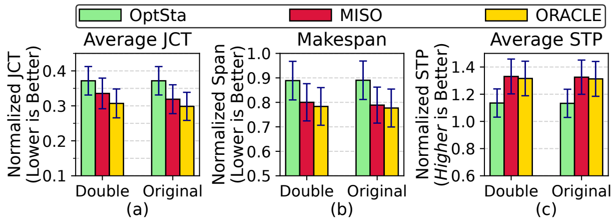

Figure 17: Sensitivity to checkpointing overhead.

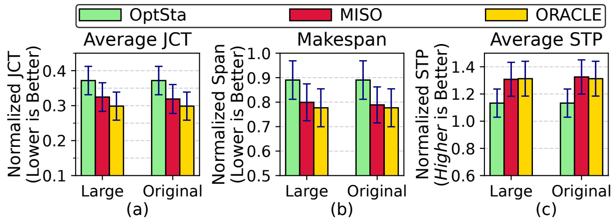

Figure 18: Sensitivity to performance prediction error.

job generation, arrivals, lengths). Hence the results yield different magnitudes of performance improvement. We use violin plots in Fig. 16 to capture this difference. For each initial condition, we normalize the measurements of all techniques over NoPart.

Our evaluation confirms that MISO indeed provides a significant improvement over competing schemes averaged over all runs, and stays close to Oracle's improvement results – not just the median/min/max, but throughout the full distribution. MISO provides about 70%, 20%, and 30% median improvement over NoPart in terms of JCT, makespan, and system throughput. These improvements are amplified in the large-scale system compared to real system evaluation, showing MISO's scalability. We have confirmed that when setting all simulation parameters to be the same as real system evaluation, they yield similar results.

Sensitivity to checkpointing overhead, performance model prediction error, and inter-arrival rate. Finally, we evaluate the sensitivity of different system and design parameters on MISO's effectiveness. Recall that MISO operation relies on (1) checkpointing, and (2) performance prediction from MPS to MIG mode. MISO incurs checkpointing overhead during MPS profiling and MIG re-partitioning. Our results (Fig. 17) confirm that this overhead does not impact MISO's benefits even when the checkpointing overhead doubles – presumably with a hypothetical system of much slower memory bandwidth or jobs that are much larger in size. Fig. 18 shows that even when the performance prediction model is just trained for a couple of epochs with a large prediction error (error from 1.7% to 9%), MISO still provides a comparable improvement over non-partitioned GPUs without a fine-tuned model.

Finally, we show that MISO remains effective as the job inter-arrival time changes (Fig. 19). This test was performed to simulate systems with different loads. A low inter-arrival time (small λ) requires MISO to profile and adjust the MIG partitions more frequently and oversubscribes the GPUs. Therefore, its relative JCT performance degrades. In spite of that, MISO still maintains its improvements in makespan and system throughput. MISO continues to provide 30% to 50% of average JCT improvement, more than 15% of makespan improvement, and more than 25% higher system throughput over NoPart across a wide range of inter-arrival rates.

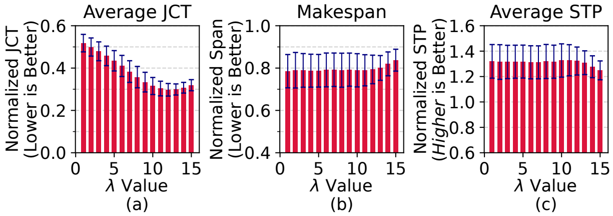

Figure 19: Sensitivity to arrival rate (λ unit: seconds).

7 RELATED WORK

Prior extensive works on co-locating workloads on CPU-based servers [49–63] do not provide a solution to GPU-specific co-location challenges (e.g., different architecture organization, resource sharing granularity, and allowable resource partitions). Consequently, multiple works have investigated GPU-specific sharing.

In particular, Clockwork [64], S^{3}DNN [65], and DART [66] design CUDA stream schedulers for multiple DNNs to share a GPU at the operator level. TimeWall [67], Gandiva [68], Gandiva-fair [69], and Antman [70] use GPU time-sharing to improve resource utilization during idle job cycles. Space sharing is suitable when a single application cannot efficiently use the entire GPU, which is addressed by Gavel [27] and Gslice [71] in MPS sharing mode. Many other works have addressed various areas in GPU sharing including communication, memory allocation, and latency sensitivity [72–76]. However, none of the above works addresses the challenges and limitations of using MIG-enabled GPU sharing which, as we discussed in Sec. 2 and 3, has its own specific challenges and benefits. Recently, Abacus [48] and Zahaf et al. [77] have used MIG-enabled GPU for their experimental evaluation, but they rely on a naive static resource partitioning – understandably so since their focus is not to improve performance via determining the best partition of resources. GPU sharing has also been studied on the device memory, including concurrent query processing [78] and virtual memory management for co-located applications [79]. These works have pushed forward the research field till the recent MIG technology appears. MISO is built upon MIG's hardware-supported memory sharing and isolation. In summary, MISO is the first work to address various challenges in operating a MIG-enabled GPU datacenter and provide solutions for improving job completion time and system throughput.

8 DISCUSSION

GPU resource partitioning beyond NVIDIA A100 GPUs. We anticipate that hardware and software support for GPU resource partitioning will become prevalent as single GPU nodes become more powerful and all the architectural resources within a single GPU cannot be maximally utilized by a single application all the time. NVIDIA's Ampere architecture (A100) is the first commercially available GPU to provide this capability via MIG technology. NVIDIA's next-generation architecture, Hopper, will continue to offer MIG support [80]. Other GPU vendors also realize this opportunity and are working toward providing similar support. For example, AMD's Compute Unit (CU) masking library in the ROCm (Radeon Open Compute) stack, will potentially allow partitioning of the CUs similar to NVIDIA's MPS [81] and similar approach is anticipated from Intel [82].

Scalability w.r.t. the number of partition combinations in future MIG-based GPUs. As GPUs evolve, it is possible that future generation GPUs may have more MIG slices, and hence, more number of combinations than today (currently, 18 combinations). There are two major implications: (1) a larger number of MIG slice types could affect MISO's performance prediction accuracy for different MIG slices, and (2) MISO's partition optimizer algorithm needs to account for a larger number of partition combinations. Fortunately, MISO's design is reasonably robust to these issues. For the first issue, our sensitivity study (Fig. 18) shows that MISO can tolerate some prediction errors in its model (from 1.7% to 9%) and still provide significant improvements. Furthermore, we can leverage transfer learning to improve the accuracy of our models as the number of combinations increases. For the second issue, we experimentally measured that Algorithm 1 finishes within 80 ms even with 10× the number of combinations (total of 180 combinations). This is because the partition optimizer runtime scales linearly with the number of combinations for a given degree of co-location. Even with a 100× increase, the optimizer finishes within a second, and its latency is overlapped with the execution of workloads.

Future work and opportunities enabled by MISO. MISO demonstrates effective co-location of multiple jobs for higher throughput. Each single GPU node can be treated as a combination of different small GPUs (i.e., multiple heterogeneous partitions within a GPU). The cloud computing providers and HPC cluster managers may expose different partitions of a large GPU directly to users as a job allocation unit. MISO will enable cloud computing users to leverage MISO's performance predictor to estimate a job's performance on different sub-GPUs, and request those partitions accordingly. *Finally, we hope that MISO can also help cloud compute providers appropiately price their sub-GPUs (in terms of monetary cost or core hours) as a single resource consumption unit and expose them as compute units for rent.*

9 CONCLUSION

In this paper, we presented MISO, a technique to leverage the MIG functionality on NVIDIA A100 GPUs to dynamically partition GPU resources among co-located jobs. MISO deploys a learning-based method to quickly find the optimal MIG partition for a given job mix running in MPS. MISO is evaluated using a variety of deep learning workloads and achieves an average job completion time that is lower than the unpartitioned GPU scheme by 49% and is within 10% of the Oracle technique.

APPENDIX

Fig. 20 visually shows all 18 possible MIG configurations in an A100 GPU. Each row represents one configuration (e.g., the second row represents (`4g.20gb`, `2g.10gb`, `1g.5gb`.)

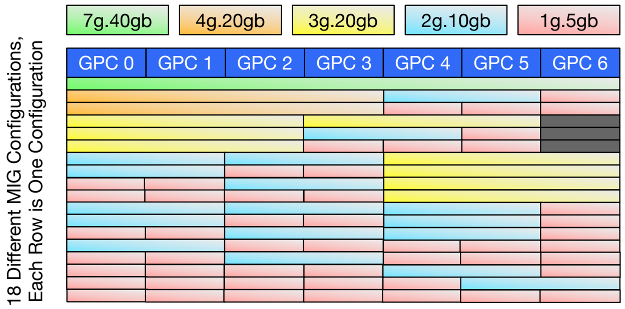

Figure 20: All possible MIG configurations in A100, created according to NVIDIA's MIG user guide [16].

ACKNOWLEDGMENTS

We would like to thank anonymous reviewers and our shepherd (Purushottam Kulkarni) for their feedback which helped us improve this paper. This research was supported by NSF Award 1910601, NSF Award 2124897, and Northeastern University. This research was also sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Justin M Wozniak, Rajeev Jain, Prasanna Balaprakash, Jonathan Ozik, Nicholson T Collier, John Bauer, Fangfang Xia, Thomas Brettin, Rick Stevens, Jamaludin Mohd-Yusof, et al. Candle/supervisor: A workflow framework for machine learning applied to cancer research. *BMC bioinformatics*, 19(18):59–69, 2018.

[2] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[3] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

[4] Victor Fung, Jiaxin Zhang, Eric Juarez, and Bobby G Sumpter. Benchmarking graph neural networks for materials chemistry. *npj Computational Materials*, 7(1):1–8, 2021.

[5] A100. NVIDIA A100 Tensor Core GPU Datasheet, 2021. URL https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf.

[6] Dong Chen, Noel Eisley, Philip Heidelberger, Sameer Kumar, Amith Mamidala, Fabrizio Petrini, Robert Senger, Yutaka Sugawara, Robert Walkup, Burkhard Steinmacher-Burow, et al. Looking under the hood of the ibm blue gene/q network. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2012.

[7] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[8] Nikoli Dryden, Roman Böhringer, Tal Ben-Nun, and Torsten Hoefler. Clairvoyant prefetching for distributed machine learning i/o. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[10] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[11] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–501. IEEE, 2020.

[12] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of {Large-Scale}{Multi-Tenant}{GPU} clusters for {DNN} training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 947–960, 2019.

[13] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960, Renton, WA, April 2022. USENIX Association. ISBN 978-1-939133-27-4. URL https://www.usenix.org/conference/nsdi22/presentation/weng.

[14] Baolin Li, Rohin Arora, Siddharth Samsi, Tirthak Patel, William Arcand, David Bestor, Chansup Byun, Rohan Basu Roy, Bill Bergeron, John Holodnak, et al. Ai-enabling workloads on large-scale gpu-accelerated system: Characterization, opportunities, and implications. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1224–1237. IEEE, 2022.

[15] MPS. NVIDIA Multi-Process Service, 2021. URL https://docs.nvidia.com/deploy/mps/.

[16] MIG. NVIDIA Multi-Instance GPU User Guide, 2021. URL https://docs.nvidia.com/datacenter/tesla/mig-user-guide/.

[17] AWS. AWS to offer NVIDIA A100 Tensor Core GPU-based Amazon EC2 instances, 2022. URL https://www.nvidia.com/en-us/data-center/a100/.

[18] Google. A2 VMs now GA—the largest GPU cloud instances with NVIDIA A100 GPUs, 2022. URL https://cloud.google.com/blog/products/compute/a2-vms-with-nvidia-a100-gpus-are-ga.

[19] Top500. Top 500 list November 2021, 2021. URL https://www.top500.org/lists/top500/2021/11/.

[20] Microsoft. Microsoft expands its AI-supercomputer lineup with general availability of the latest 80GB NVIDIA A100 GPUs in Azure, 2021. URL https://azure.microsoft.com/en-us/blog/microsoft-expands-its-aisupercomputer-lineup-with-general-availability-of-the-latest-80gb-nvidia-a100-gpus-in-azure-claims/.

[21] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[22] Baolin Li, Viiay Gadepally, Siddharth Samsi, and Devesh Tiwari. Characterizing multi-instance gpu for machine learning workloads. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 724–731. IEEE, 2022.

[23] Stijn Eyerman and Lieven Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE micro*, 28(3):42–53, 2008.

[24] Stijn Eyerman, Pierre Michaud, and Wouter Rogiest. Multiprogram throughput metrics: A systematic approach. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(3):1–26, 2014.

[25] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–14, 2018.

[26] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A {GPU} cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 485–500, 2019.

[27] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. {Heterogeneity-Aware} cluster scheduling policies for deep learning workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 481–498, 2020.

[28] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 127–144. ACM, 2014.

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[30] Mark Veillette, Siddharth Samsi, and Chris Mattioli. Sevir: A storm event imagery dataset for deep learning applications in radar and satellite meteorology. *Advances in Neural Information Processing Systems*, 33:22009–22019, 2020.

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020.

[33] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[34] NVIDIA-A100. NVIDIA A100 TENSOR CORE GPU, 2022. URL https://www.nvidia.com/en-us/data-center/a100/.

[35] Amy Ousterhout, Jonathan Perry, Hari Balakrishnan, and Petr Lapukhov. Flexplane: An experimentation platform for resource management in datacenters. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 438–451, 2017.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[37] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[38] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[40] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR, 2016.

[41] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[42] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[43] Woong Shin, Vladyslav Oles, Ahmad Maroof Karimi, J Austin Ellis, and Feiyi Wang. Revealing power, energy and thermal dynamics of a 200pf pre-exascale supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.

[44] Xingfu Wu, Valerie Taylor, Justin M Wozniak, Rick Stevens, Thomas Brettin, and Fangfang Xia. Performance, energy, and scalability analysis and improvement of parallel cancer deep learning candle benchmarks. In *Proceedings of the 48th International Conference on Parallel Processing*, pages 1–11, 2019.

[45] Siddharth Samsi, Matthew L Weiss, David Bestor, Baolin Li, Michael Jones, Albert Reuther, Daniel Edelman, William Arcand, Chansup Byun, John Holodnack, et al. The mit supercloud dataset. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8. IEEE, 2021.

[46] Hugging-Face. Hugging Face: The AI community building the future., 2022. URL https://huggingface.co/.

[47] Keras. keras-io., 2022. URL https://github.com/keras-team/keras-io.

[48] Weihao Cui, Han Zhao, Quan Chen, Ningxin Zheng, Jingwen Leng, Jieru Zhao, Zhuo Song, Tao Ma, Yong Yang, Chao Li, et al. Enable simultaneous dnn services based on deterministic operator overlap and precise latency prediction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[49] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Notices*, 48 (4):77–88, 2013.

[50] Yunqi Zhang, Michael A Laurenzano, Jason Mars, and Lingjia Tang. SMiTe: Precise QoS Prediction on Real-System SMT Processors to Improve Utilization in Warehouse Scale Computers. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 406–418. IEEE, 2014.

[51] Sergey Blagodurov, Alexandra Fedorova, Evgeny Vinnik, Tyler Dwyer, and Fabien Hermenier. Multi-Objective Job Placement in Clusters. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2015.

[52] Haishan Zhu and Mattan Erez. Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems. *ACM SIGARCH Computer Architecture News*, 44(2):33–47, 2016.

[53] Harshad Kasture, Davide B Bartolini, Nathan Beckmann, and Daniel Sanchez. Rubik: Fast Analytical Power Management for Latency-Critical Systems. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 598–610. IEEE, 2015.

[54] Xiaodong Wang, Shuang Chen, Jeff Setter, and José F Martínez. SWAP: Effective Fine-Grain Management of Shared Last-Level Caches with Minimum Hardware Support. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 121–132. IEEE, 2017.

[55] Harshad Kasture and Daniel Sanchez. Ubik: Efficient Cache Sharing with Strict QoS for Latency-Critical Workloads. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 729–742. ACM, 2014.

[56] Nosayba El-Sayed, Anurag Mukkara, Po-An Tsai, Harshad Kasture, Xiaosong Ma, and Daniel Sanchez. KPart: A Hybrid Cache Partitioning-Sharing Technique for Commodity Multicores. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 104–117. IEEE, 2018.

[57] Yaocheng Xiang, Xiaolin Wang, Zihui Huang, Zeyu Wang, Yingwei Luo, and Zhenlin Wang. DCAPS: Dynamic Cache Allocation with Partial Sharing. In *Proceedings of the Thirteenth EuroSys Conference*, page 13. ACM, 2018.

[58] Cong Xu, Karthick Rajamani, Alexandre Ferreira, Wesley Felter, Juan Rubio, and Yang Li. dCat: Dynamic Cache Management for Efficient, Performance-Sensitive Infrastructure-as-a-Service. In *Proceedings of the Thirteenth EuroSys Conference*, page 14. ACM, 2018.

[59] Jinsu Park, Seongbeom Park, Myeonggyun Han, Jihoon Hyun, and Woongki Baek. HyPart: A Hybrid Technique for Practical Memory Bandwidth Partitioning on Commodity Servers. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, page 5. ACM, 2018.

[60] Jinsu Park, Seongbeom Park, and Woongki Baek. CoPart: Coordinated Partitioning of Last-Level Cache and Memory Bandwidth for Fairness-Aware Workload Consolidation on Commodity Servers. In *Proceedings of the Fourteenth EuroSys Conference 2019*, page 10. ACM, 2019.

[61] Artemiy Margaritov, Siddharth Gupta, Rekai Gonzalez-Alberquilla, and Boris Grot. Stretch: Balancing QoS and Throughput for Colocated Server Workloads on SMT Cores. In *2019 IEEE International Symposium*

on High Performance Computer Architecture (HPCA), pages 15–27. IEEE, 2019.

[62] Tirthak Patel and Devesh Tiwari. CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical jobs for Warehouse Scale Computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 193–206. IEEE, 2020.

[63] Jifei Yi, Benchao Dong, Mingkai Dong, Ruizhe Tong, and Haibo Chen. Mt2: Memory bandwidth regulation on hybrid nvm/dram platforms. In *20th USENIX Conference on File and Storage Technologies (FAST 22), Santa Clara, CA*, 2022.

[64] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving {DNNs} like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 443–462, 2020.

[65] Husheng Zhou, Soroush Bateni, and Cong Liu. Sˆ 3dnn: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 190–201. IEEE, 2018.

[66] Yecheng Xiang and Hyoseung Kim. Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 392–405. IEEE, 2019.

[67] Tanya Amert, Zelin Tong, Sergey Voronov, Joshua Bakita, F Donelson Smith, and James H Anderson. Timewall: Enabling time partitioning for real-time multicore+ accelerator platforms. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 455–468. IEEE, 2021.

[68] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 595–610, 2018.

[69] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[70] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. {AntMan}: Dynamic scaling on {GPU} clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 533–548, 2020.

[71] Aditya Dhakal, Sameer G Kulkarni, and KK Ramakrishnan. Gslice: controlled spatial sharing of gpus for a scalable inference platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 492–506, 2020.

[72] Munkyu Lee, Hyunho Ahn, Cheol-Ho Hong, and Dimitrios S Nikolopoulos. gshare: A centralized gpu memory management framework to enable gpu memory sharing for containers. *Future Generation Computer Systems*, 130:181–192, 2022.

[73] Kiran Ranganath, Joshua D Suetterlein, Joseph B Manzano, Shuaiwen Leon Song, and Daniel Wong. Mapa: Multi-accelerator pattern allocation policy for multi-tenant gpu servers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.

[74] Wei Zhang, Quan Chen, Kaihua Fu, Ningxin Zheng, Zhiyi Huang, Jingwen Leng, and Minyi Guo. *Astraea: Towards QoS-Aware and Resource-Efficient Multi-Stage GPU Services*, page 570–582. Association for Computing Machinery, New York, NY, USA, 2022. ISBN 9781450392051. URL https://doi.org/10.1145/3503222.3507721.

[75] Sina Darabi, Negin Mahani, Hazhir Baxishi, Ehsan Yousefzadeh-Asl-Miandoab, Mohammad Sadrosadati, and Hamid Sarbazi-Azad. Nura: A framework for supporting non-uniform resource accesses in gpus. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(1), feb 2022. doi: 10.1145/3508036. URL https://doi.org/10.1145/3508036.

[76] Chao Chen, Chris Porter, and Santosh Pande. Case: A compiler-assisted scheduling framework for multi-gpu systems. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '22, page 17–31, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392044. doi: 10.1145/3503221.3508423. URL https://doi.org/10.1145/3503221.3508423.

[77] Houssam-Eddine Zahaf, Ignacio Sanudo Olmedo, Jayati Singh, Nicola Capodieci, and Sebastien Faucou. Contention-aware gpu partitioning and task-to-partition allocation for real-time workloads. In *29th International Conference on Real-Time Networks and Systems*, pages 226–236, 2021.

[78] Kaibo Wang, Kai Zhang, Yuan Yuan, Siyuan Ma, Rubao Lee, Xiaoning Ding, and Xiaodong Zhang. Concurrent analytical query processing with gpus. *Proceedings of the VLDB Endowment*, 7(11):1011–1022, 2014.

[79] Kaibo Wang, Xiaoning Ding, Rubao Lee, Shinpei Kato, and Xiaodong Zhang. Gdm: Device memory management for gpgpu computing. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):533–545, 2014.

[80] NVIDIA. NVIDIA Hopper GPU Architecture, 2022. URL https://www.nvidia.com/en-us/technologies/hopper-architecture/.

[81] Nathan Otterness and James H Anderson. Exploring amd gpu scheduling details by experimenting with "worst practices". In *29th International Conference on Real-Time Networks and Systems*, pages 24–34, 2021.

[82] Intel. OneAPI GPU Optimization Guide, 2022. URL https://www.intel.com/content/dam/develop/external/us/en/documents/oneapi-gpu-optimization-guide.pdf.