Using Multi-Instance GPU for Efficient Operation of Multi-Tenant GPU Clusters

Baolin Li  
Northeastern University

Tirthak Patel  
Northeastern University

Vijay Gadepally  
MIT

Siddharth Samsi  
MIT

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. We propose MISO, a technique to exploit the Multi-Instance GPU (MIG) capability of NVIDIA A100 GPUs 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.

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 [10–13] 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 [14, 15]. 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).

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 offering, industrial research computing clusters, and academic HPC centers [16–19] and. 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 cost of renting GPU resources or operating HPC clusters. Our proposed solution, MISO, will be made publicly available as an open-source package for HPC centers and researchers to reproduce and refine.

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 improving 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 most 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 GPU partitioning.
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.

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 [20]. 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 soon. Along with these developments, MISO will continue to benefit current and future MIG-enabled cloud computing data centers and HPC centers [16–19].

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.

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

### Table 1: Complete list of MIG profile on an A100 GPU [15].

| 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         |
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, 2gb and 3g, 2gb cannot co-exist in a single A100. In total, there are 18 MIG configurations on an A100 GPU [15]. 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 MISO, we use **system throughput**, or STP to measure the combined progress rate of all jobs [21, 22]. 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 CPU 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:

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

This particular definition and similar variants have been widely used in the literature for denoting system throughput [21, 22].

**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 previous works [23–25], 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 (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 (top 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 (bottom bar) can provide higher system throughput than the MPS co-location. However, our MPS co-location shared the resources equally (33% each) 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 (the middle bar) and noticed that the MIG partition still yields higher performance. This is because MIG provides better performance isolation and resource exclusivity among co-located jobs than MPS (illustrated in Fig. 1) [15]. We note that while it is possible that some MPS configurations can outperform certain MIG configurations, our motivational example serves the simple purpose of demonstrating that better isolation can lead to higher performance.

**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.
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 MPS 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).

4 MISO: THE MIG SOLUTION

Fig. 5 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. 6.

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, 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 together 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 incurs up to 8x 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 [25, 26] 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.

**Predictor Design.** We construct a variant of the U-Net [27] convolutional autoencoder model. It is a lightweight model with fewer encoder/decoder blocks and fewer convolutional filters compared to typical models used in applications [28]. As shown in Fig. 6, 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 input to the model is a 3x7 matrix collected from MPS, corresponding to 3 MPS levels and 7 jobs running concurrently. The output from the model is also a 3x7 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 (coefficient of determination) of 0.96.

**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 3x7 MPS matrix input (with dummy filling) and a 3x7 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 [29]. These hyperparameters along with others such as learning rate and activation function are selected using the ASHA hyperparameter tuning algorithm [30] on Ray Tune [31]. 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.

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, we 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 \( \mathbf{x} = [x_1, x_2, ..., x_m] \), where \( x_i \) 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, 24]. 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 \( \mathbf{x} \). Without the job speedup information on each slice type, this is a black-box optimization problem: we cannot know the performance of each configuration \( \mathbf{x} \) 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. 7 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 \( f_i : x_i \rightarrow k_i \), 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.

![Figure 7: MISO optimizer to generate optimal MIG partition.](image)

on 3g. 20gb compared to the full GPU, \( f_1(3) = 0.5 \). The problem is as follows:

\[
\max_{\mathbf{x}} \sum_{i=1}^{m} f_i(x_i) \tag{2}
\]

\[
\text{s.t. } \mathbf{x} \in P_{mig} \tag{3}
\]

\[
\|\mathbf{x}\|_0 = m \tag{4}
\]

Here \( P_{mig} \) represents all the available partition configurations on an A100 GPU (from MIG documentation [15]). For example, \( \mathbf{x} = [4, 2, 1] \) 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 [15], 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.

The other major source 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;
When a new job is scheduled, the host GPU needs to go into the excessive I/O and runtime overhead often offsetting the gains from different MIG slices provide different GPU nvidia-smi monitors the GPU memory usage for individual jobs using the memory needed for each job. During the MPS stage, running on smaller slices. Users may specify the minimum GPU memory sizes; some jobs may face out-of-memory errors when.

Upon the profiling completion, the process repeats as the GPU in MIG mode will need to be checkpoint-restarted to run on MPS. MISO’s partition 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 keeps an MPS control daemon in the background, which sets the MPS levels for all jobs during MISO profiling (by the CUDA_MPS_ACTIVE_THREAD_PERCENTAGE environment variable). 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 have integrated these commands into Python function calls using the subprocess module. These MPS/MIG APIs will be open-sourced as our contributions to the community.

MISO’s server API (Fig. 5) 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 [32]. 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

Algorithm 1: MISO’s partition optimizer.

```
best_obj ← 0 // Maximum objective so far
best_config ← None // Best partition so far
P_valid ← list of P_mig partitions whose length equals m

foreach \( x \) in P_valid do
    obj_func ← \( \sum_{i=1}^{m} f(x_i) \)
    if obj_func > best_obj then
        best_obj ← obj_func
        best_config ← \( x \)
    end
end
return best_config
```
Table 2: Workloads used to evaluate MISO.

| Model          | Batch Sizes | Application                      |
|----------------|-------------|----------------------------------|
| ResNet50 [34]  | 64, 128, 256, 512 | Image classification with residual learning |
| MobileNet [35] | 64, 128, 256, 512 | Image classification on light weight model |
| BERT [36]      | 2, 4, 6, 8   | Sentiment analysis of the IMDB movie reviews |
| Transformer [37]| 16, 32, 64, 128 | Time series prediction of engine noise measurement |
| DeepSpeech [38]| 2, 4, 8, 16   | Automatic speech recognition of the LJSpeech dataset |
| Embedding [39] | 64, 128, 256, 512 | Word embedding model for message topic classification |
| Graph NN [20]  | 64, 128, 256, 512 | Property prediction of quantum chemistry molecular graphs |
| CycleGAN [40]  | 1, 2, 3, 4    | Learning of mapping for image-to-image translation |

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 for each trial and use a 1000-job mix for each trial and use a limit helps us guarantee the completion of one set of experiments within a day. The job arrival follows a Poisson distribution with a parameter is also swept over a range of values to 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.

For simulator evaluations, we use the same trace to generate a 1000-job mix for each trial and use a 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 [41–43]. We uniformly sample the DL model and training batch size from Table 2. These workloads come from Hugging Face [44] and the keras . io repository [45]. 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 [46] 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).

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.

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. 8 shows the average job completion time for different competing strategies, normalized to unpartitioned (NoPart) strategy. Recall that unpartitioned strategy does not create MIG instances to colocate 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. 8(a)). Recall that OptSta determines the optimal MIG instance partitioning via an exhaustive offline search process. The same partition is assigned 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. 8(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. 8 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. 9 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. 8(a)) is not a result of MISO’s aggressive attention to certain jobs. In fact, Fig. 9 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%. Next, we dig deeper to understand the reason behind MISO’s strength.

Why does MISO perform effectively? Fig. 10 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 colocaction-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. Recall 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. 10(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.

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.
with different initial conditions (different job generation, arrivals, and system loads). Our evaluation confirms that MISO’s operation relies on (1) checkpointing overhead, performance model prediction error, and (2) performance prediction from MPS to MIG mode. MISO incurs checkpointing overhead during MPS profiling and MIG re-partitioning. Our results (Fig. 12) confirm that this overhead does not impact MISO’s benefit 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. 13 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. This test was performed to simulate systems with different loads. A low inter-arrival time (small $\lambda$) 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 is still able to maintain 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.

### 7 RELATED WORK

There have been extensive works in co-locating workloads on CPU-based servers with managed and unmanaged (explicit division and assignment of resources) sharing of core, cache, memory, network, and I/O resources [47–61]. While these works are effective on traditional CPUs, they 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 resource sharing.

In particular, Clockwork [62], S$^3$DNN [63], and DART [64] design CUDA stream schedulers for multiple DNNs to share a GPU at the operator level. TimeWall [65], Gandiva [66], Gandiva-fair [67], and Antman [68] 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 [25] and Galice [69] in MPS sharing mode. Many other works have addressed various areas in GPU sharing including communication, memory allocation, and latency sensitivity [70–75]. 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 [46] and Zahaf et al. [76] 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. 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 & system throughput via a novel MPS-to-MIG performance estimation to mitigate MIG-specific overhead of configuration exploration.
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 achieves an average job completion time that is lower than the unpartitioned GPU scheme by 49% and is within 10% of the Oracle technique.

ACKNOWLEDGMENTS

This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001, and United States Air Force Research Laboratory Cooperative Agreement Number FA8750-19-2-1000. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering, or the United States Air Force. 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, Jalamadhin Mohd-Yusof, et al. CANDLE/supervisor: A workflow framework for machine learning applied to cancer research. BMC bioinformatics, 19(1):59–69, 2018.

[2] Maziar Raissi, Paris Perdikaris, and George 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, Perm Buttaglia, and Jean-Roch Vlimant. Graph neural networks applied to cancer research. IEEE Machine Learning: Science and Technology, 2(2):0001, 2020.

[4] Victor Fang, 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-1754950-xr-web.pdf.

[6] Dong Chen, Noel Esley, Philip Heidelberger, Sameer Kumar, Amin Maimudala, Fabrizio Petrin, Robert Sengr, 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 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.

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

[9] 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.

[10] Yuqiang Peng, Xinyi Bao, Yangru 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.

[11] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongje Jeon, Junqing 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.

[12] 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.

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

[14] 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.

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

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

[17] Liam Li, Kevin Jamieson, Ali Hassanzadeh, Ekaterina Gonina, Jonathan Ben-Tiru, Moritz Hardt, Ben Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. Proceedings of Machine Learning and Systems, 2:230–246, 2020.

[18] 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.

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

[20] Amy Ousterhout, Jonathan Perry, Hari Balakrishnan, and Peti Lapshov. 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.

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

[22] 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.

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

[24] Amy Ousterhout, Jonathan Perry, Hari Balakrishnan, and Peti Lapshov. 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.

[25] Liam Li, Kevin Jamieson, Ali Hassanzadeh, Ekaterina Gonina, Jonathan Ben-Tiru, Moritz Hardt, Ben Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. Proceedings of Machine Learning and Systems, 2:230–246, 2020.

[26] 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.

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

[28] Amy Ousterhout, Jonathan Perry, Hari Balakrishnan, and Peti Lapshov. 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.

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

[30] 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.

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

[32] Amy Ousterhout, Jonathan Perry, Hari Balakrishnan, and Peti Lapshov. 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.
