







# Improving Cluster Utilization Through Adaptive Resource Management for Deep Neural Network and CPU Jobs Colocation

Han Zhao , Weihao Cui , Quan Chen , *Member, IEEE*, Jingwen Leng , *Member, IEEE*, Deze Zeng , *Member, IEEE*, and Minyi Guo , *Fellow, IEEE*

**Abstract**—While deep neural network (DNN) models are mainly trained using GPUs, many companies and research institutions build shared GPU clusters. These clusters host DNN training jobs, DNN inference jobs, and CPU jobs (jobs in traditional areas). DNN training jobs require GPU for main computation and CPU for auxiliary computation. Some DNN inference jobs could rely solely on CPU, while others must utilize both CPU and GPU. Our investigation demonstrates that the number of cores allocated to a training job significantly impacts its performance, and that DNN inference jobs can make use of the limited CPU cores on the GPU nodes. To accomplish this, we characterize representative deep learning models in terms of their CPU core requirements for their training jobs and inference jobs, and investigate their sensitivity to other CPU-side resource contention. Based on the characterization, we propose SODA, a scheduling system comprised of an adaptive CPU allocator, a multi-array job scheduler, a hardware-aware inference job placer, and a real-time contention eliminator. The experimental results indicate that SODA increases GPU utilization by an average of 19.9%, while maintaining the quality of service target for all DNN inference jobs and the queuing performance of CPU jobs.

**Index Terms**—DNN training, DNN inference, schedule.

## I. INTRODUCTION

THE training of DNN models [1], [2], [3] is well-known to be complex and time-consuming. To resolve this challenge, a growing number of businesses establish GPU clusters and share them across multiple departments to amortize the expense. This leads to the emergence of the multi-tenant GPU cluster, as shown in Fig. 1. In the cluster, GPU nodes provide both GPU and CPU resources. The jobs could then be categorized as GPU jobs and CPU jobs. GPU jobs require both GPU and CPU resources, whereas CPU jobs demand CPU resources solely.

Manuscript received 9 October 2022; revised 27 May 2023; accepted 4 August 2023. Date of publication 10 August 2023; date of current version 8 November 2023. This work was supported by the National Natural Science Foundation of China under Grants 62022057, 61832006, and 61872240. Recommended for acceptance by J. Zhai. (*Corresponding authors: Quan Chen; Minyi Guo.*)

Han Zhao, Weihao Cui, Quan Chen, Jingwen Leng, and Minyi Guo are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: chen-quan@cs.sjtu.edu.cn; guo-my@cs.sjtu.edu.cn).

Deze Zeng is with the School of Computer Science, China University of Geosciences, Wuhan 430074, China.

Digital Object Identifier 10.1109/TC.2023.3303988

Specifically, GPU jobs are mainly DNN jobs, including DNN training jobs and DNN inference jobs. DNN training jobs conduct forward and backward computations over many iterations. They involve frequent CPU-GPU interactions. DNN inference jobs serve user queries. They are launched and terminated based on the query load [4], [5]. Since inference jobs only require forward computation, some models could rely solely on CPU [6] while others need both CPU and GPU [7], [8]. As a result, this multi-tenant GPU cluster represents a new private cloud paradigm. This new paradigm requires a detailed study of workload characteristics, resource utilization, and optimization.

There have been several studies that focus on the characterization and scheduling of DNN training jobs, such as Microsoft's [9] and Google's [10]. Jeon et al. [9] pay more attention to DNN training jobs' GPU affinity, error, and queuing time. It improves the training job's performance by maximizing the GPU affinity. KERP [10] demonstrates that 16% of nodes in Google's cluster have peak bandwidth more than 70% of available bandwidth. It proposes better memory bandwidth allocation when colocating DNN training jobs and CPU jobs. Although these two works consider GPU affinity and bandwidth contention, they overlook the DNN model's CPU requirement. The CPU core allocation greatly impacts the training job's performance. Previous works' naive CPU allocation results in poor GPU utilization and low throughput. Worse, they do not explore all the possible resource interference, and do not consider the possibility of using CPU cores on GPU nodes for DNN inference jobs.

In this work, we conduct a detailed investigation of the DNN job's resource requirements and sensitivity to multiple resource contention on a production-scale GPU cluster. Four artificial intelligence startup companies and one research institution share this cluster. Three key findings emerge from the investigation. First, there are more than 30,000 DNN training jobs over one month. A large number of training jobs apply for one core or two cores, and a considerable number of training jobs apply for over ten cores. Regardless of the notable application mode, our analysis reveals that DNN training jobs have different needs for the cores to achieve the best performance. The CPU cores significantly impact the DNN training job's performance (Section IV).

Second, the widely-used scheduling algorithms, such as FIFO [11] and DRF [12], schedule the CPU jobs and GPU jobs uniformly. Naive resource allocation brings the GPU

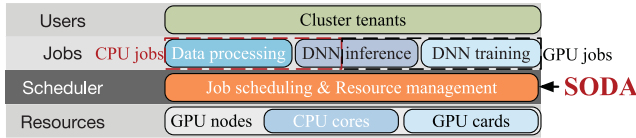


Fig. 1. The target multi-tenant GPU cluster.

fragmentation due to the insufficient CPU core on the GPU nodes. Additionally, GPU jobs that apply to one or two GPUs may result in GPU fragmentation for the GPU jobs applying to four or more GPUs. These two fragmentation cases contribute to the long queuing time of GPU jobs.

Third, DNN inference jobs all rely on GPU for computation. When the query load is low at night, some DNN inference jobs also need to reside on the GPU. Existing scheduling systems do not perceive the possibility of using CPU cores on GPU nodes to serve inference jobs. While the user queries exhibit a diurnal pattern [13], the few user queries at night mean low GPU utilization.

The three findings motivate us to propose **SODA**, a job scheduling system that improves cluster utilization by adaptive resource management. SODA targets the multi-tenant GPU cluster, which colocates DNN jobs and CPU jobs. Four challenges have to be resolved in SODA. 1) The best-fit number of CPU cores required for DNN training jobs varies. SODA must determine which is the ideal one at runtime. 2) Existing scheduling algorithms' unified job scheduling brings many resource contention. SODA must first resolve the GPU fragmentation produced by these contentions, then eliminate the performance degradation of DNN jobs. 3) It is unknown whether the DNN inference job could use the CPU cores on GPU nodes. SODA needs to explore the possibility of using the limited CPU resources on GPU nodes and coordinate the usage of GPU and CPU at runtime. 4) the CPU-side computation of DNN training jobs contends for the shared resources (e.g., last-level cache and memory bandwidth), and the contention may result in severe performance degradation. SODA has to be able to monitor the contention on shared resources and schedule the training jobs accordingly to avoid serious contention.

To be more specific, SODA consists of an *adaptive CPU allocator*, a *multi-array job scheduler*, a *hardware-aware inference job placer*, and a *real-time contention eliminator*. The adaptive CPU allocator finds the best-fit CPU core number for a DNN training job. Based on the model type information, the allocator finds the optimal CPU number in less than four attempts. The multi-array job scheduler differentiates between CPU and GPU jobs and schedules them accordingly. Additionally, the scheduler divides the CPU resources into multiple arrays, each corresponding to one job array. When many jobs in one job array are queued and the other job array is idle, it could preempt some resources accordingly. The inference job placer can identify the possibility of using the CPU cores on GPU nodes to serve the inference job. It coordinates the use of CPU and GPU in the runtime to reduce the unnecessary GPU occupation. The contention eliminator monitors the bandwidth used by each CPU job. When a CPU job's bandwidth utilization surpasses a specified threshold, the eliminator throttles its bandwidth usage to avoid performance impact on GPU jobs.

This paper makes the following main contributions:

- **Comprehensive analysis of CPU-side resource requirements of DNN training jobs.** The in-depth analysis enables the design of the SODA for maximizing the performance of DNN training jobs.
- **The design of an adaptive CPU allocation algorithm.** It identifies the best-fit CPU core number that should be allocated to a DNN training job.
- **The design of a multi-array job scheduling policy.** Based on the CPU requirements of DNN jobs and the shared resource contention, the multi-array scheduling reduces the GPU fragmentation.
- **The design of a hardware-aware inference job placement strategy.** It identifies the possibility of using the GPU node's limited CPU resources for DNN inference jobs, and coordinates the use of GPU and CPU at runtime.

Experimental results based on real-system job trace show that SODA improves the GPU utilization by 19.9%, while all the DNN inference jobs could provide the service within the quality-of-service target and the queuing performance of CPU jobs does not get degradation.

## II. RELATED WORK

### A. Scheduling Algorithm

Previous research provided a variety of scheduling algorithms for resolving the cluster job scheduling problem. DRF [12], a generalization of max-min fairness to multiple resource types, addressed the problem of fair resource allocation to users with varying demands. All mainstream job scheduling frameworks, such as Yarn [14] and Mesos [15], involve it as a scheduling option. Choosy [16] is an extension of max-min fairness that addressed the problem of max-min fairness under constraint conditions. Delay scheduling [17] is a straightforward approach in which, when the scheduled job is unable to launch a local task, it waits a brief period of time, allowing other jobs to start tasks instead. This method is also utilized and adapted to improve throughput in many job scheduling frameworks. These works are orthogonal to ours, as ours focuses on the design of the cluster's scheduling system rather than the scheduling method.

### B. Scheduling System

Numerous research works [13], [18], [19], [20], [21] studied the scheduling problem in a variety of scenarios. Gandiva [22] analyzed the application's characteristics by investigating the application's training characteristics and determining the optimal hyper-parameter setting for the Auto-ML scenario. Kelp [10] sought to regulate memory bandwidth allocation on the CPU side, hence lowering associated interference and increasing the cluster's scheduling efficiency. Optimus [18] reduced training time by utilizing online performance models to determine the optimal resource configuration. Baymax [13] and Prophet [19] utilized multitasking to maximize the usage of the cluster for pure GPU workloads. Since these efforts are targeted at certain scenarios, they are inappropriate for multi-tenant GPU clusters.

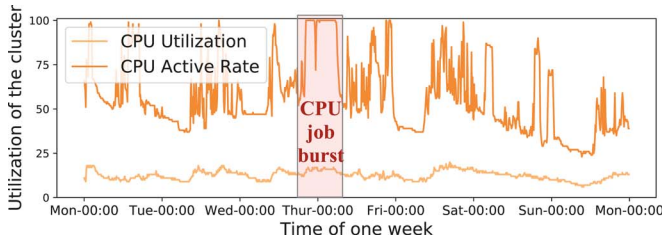


Fig. 2. The CPU and GPU utilization trend of the cluster through one week.

Many works focus on efficient array management to optimize cluster scheduling. Both Yarn [14] and Borg [23] support multi-array scheduling. The central scheduler could adjust the array's resource share based on the job load. Besides, Mercury [24] and Apollo [25] assign each array a scheduler, which brings better independence and scalability. However, these works' resource partition is at the granularity of nodes. Therefore, they also suffer from the same resource contention as our baseline system when DNN jobs and CPU jobs are in the same scheduling array. If two jobs are in different arrays, the CPU cores on GPU nodes will be wasted. Furthermore, since they are unaware of DNN jobs' resource requirements, they cannot optimize the resource management in the GPU cluster.

### III. MOTIVATION

In this section, we first present our findings based on the comprehensive analysis of a production-scale GPU cluster. The cluster has two main problems: low resource utilization and long job queuing time. Then, we locate the root causes behind the cluster's problems.

#### A. Real-World GPU Cluster Investigation

We conduct our investigation on a real-world multi-tenant GPU cluster (A GPU cluster from AISpeech Co., Ltd.). The cluster consists of around 80 multi-GPU servers (mainly 1080Ti). Each server has two sockets and is primarily powered by Intel Xeon Gold 6132 CPUs with 14 cores. Four artificial intelligence startup companies and one research institution share this cluster. These companies specialize in automatic speech recognition [26], natural language processing (NLP) [2], and computer vision (CV) [1].

The cluster is managed by a centralized job scheduling system, SLURM [27], which schedules jobs from different groups using FIFO [11]. Each job can request a specific number of CPUs and GPUs. **All GPU jobs on this cluster are DNN jobs. CPU jobs are traditional machine learning jobs and data processing jobs.** We collect the cluster's CPU/GPU usage characteristics and job information over a week, which leads to a contradicting observation.

**Resource Usage:** The resource usage includes two metrics: active rate and utilization rate. The active CPU rate is defined in Equation (1) as the ratio of actively using CPUs to the total number of CPUs. The utilization rate of an active CPU reflects how much time it spends doing actual work and is collected by the operating system. We calculate the cluster's CPU utilization

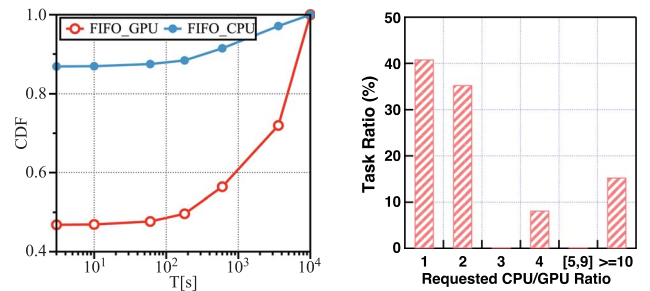
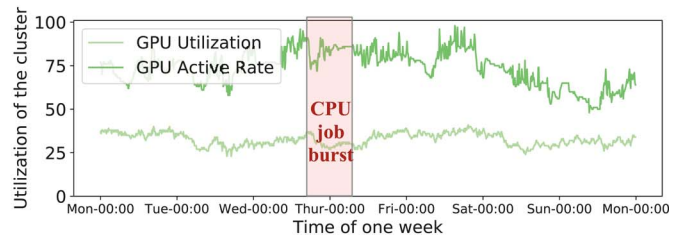


Fig. 3. Information of DNN training jobs.

rate as the average across all active CPUs. The GPU-related metrics are calculated similarly.

#### Active CPU Rate

$$= \text{Number of Active CPUs} / \text{Total CPUs} \quad (1)$$

We collect the cluster's CPU and GPU usage in a week. Fig. 2 shows that the GPU utilization is consistently higher than the CPU, which can be explained by the high computation requirement of training jobs. Besides, the GPU exhibits a relatively stable active rate, which does not exceed 80% over 90% of the time. On the other hand, the CPU active rate exhibits a diurnal pattern and reach 100% in peak times.

**Queueing Delay:** We now analyze the job queuing delay in Fig. 3(a). We observe that GPU jobs suffer from longer queuing time than CPU jobs. About 48.1% of GPU jobs wait for at least 3 minutes, and 41.3% of GPU jobs wait for more than 10 minutes. However, Fig. 2 shows that the GPU active rate rarely exceeds 80%, and the CPU active rate only reaches 100% at limited times. This leads to a contradicting discovery. While GPU jobs do not fully utilize GPU resources, many jobs wait for a long time. We then dive into the jobs' resource application for the root causes.

#### B. Root Causes

GPU jobs suffer from queuing because no GPU node could meet their resource requirement. Since the scheduling system schedules GPU jobs and CPU jobs uniformly, CPU jobs may contend the CPU resource on the GPU node with GPU jobs. As shown in Fig. 2, there is a CPU job burst on Wednesday, and the CPU active rate easily reaches 100%. Therefore, the CPU occupation of CPU jobs leads to the GPU fragmentation. **The first reason for GPU jobs' queuing is the CPU resource contention introduced by CPU jobs.**



Second, GPU jobs may contend CPU resources with GPU jobs. We then study the CPU-GPU ratio to check whether GPU jobs also introduce the CPU resource contention. Fig. 3(b) shows that 15.3% of GPU jobs request more than 10 CPU cores, and 76.1% of GPU jobs request 1 CPU or 2 CPUs. Theoretically, excessive CPU resource allocation leads to insufficient CPU cores of the GPU nodes. These GPU nodes then cannot accommodate the incoming GPU jobs. **Therefore, GPU jobs suffer from queuing due to the CPU resource contention between GPU jobs. Furthermore, insufficient CPU resource allocation causes the slowdown of jobs, which also aggravates queuing.**

Third, there may be GPU resource contention between GPU jobs with different configurations. For example, when many jobs apply for 1 GPU or 2 GPUs, naive GPU allocation easily leads to fragmentation of the GPU nodes. When the jobs applying for 4 GPUs arrive, these fragmented nodes could not serve these jobs. Therefore, unified job scheduling leads to the GPU fragmentation introduced by GPU jobs.

Fourth, there are some unreasonable GPU occupations of DNN inference jobs. In order to satisfy the quality-of-service target, DNN inference jobs generally choose GPU resources for computation. These jobs are launched and terminated due to the query load. However, we observe that 12 inference jobs always reside on the GPUs, even though there are few user queries at night. These twelve inference jobs correspond to twelve user-facing services on the cluster.

Finally, there are also implicit resource contentions in the GPU node, such as memory, network, and PCIe contention. These contentions may also bring GPU jobs' performance degradation, which further exacerbates GPU jobs' queuing.

**Therefore, we can summarize the four problems that the GPU clusters encounter.** (1) Unreasonable CPU requirements of GPU jobs bring the CPU resource contention, which leads to GPU fragmentation and further GPU jobs' queuing. (2) The unified scheduling of CPU jobs and GPU jobs also brings the CPU contention and GPU fragmentation. (3) Existing scheduling systems do not perceive the possibility of using CPU cores on GPU nodes for DNN inference jobs. (4) GPU jobs may encounter performance degradation due to the implicit resource contention on the CPU side, which could exacerbate the GPU jobs' queuing.

#### IV. ANALYSIS OF RESOURCE REQUIREMENT

Since this work targets the GPU cluster that GPU jobs are mainly DNN jobs, we first analyze the CPU-side resource requirements of mainstream models' training jobs, and then explore the possibility of using the GPU node's CPU cores for inference jobs.

As for DNN training jobs, we seek to answer three questions. (1) How does the CPU core number affect the job's performance and What is the best-fit CPU number for a training job? (2) What are the factors that determine the CPU demands of the DNN training job and how the factors determine its demand? (3) Is there any other resource contention that impacts the DNN training job's performance?

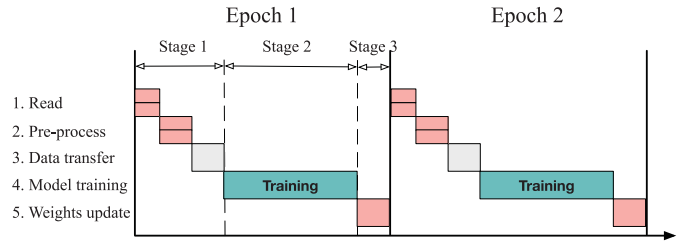


Fig. 4. The CPU-GPU collaborative process.

As for DNN inference jobs, we seek to answer two questions. (1) How to determine whether a DNN model's inference job could rely on the GPU node's CPUs? (2) Is there performance interference between the DNN inference jobs using CPU and the DNN training jobs using GPU?

#### A. CPU-GPU Collaborative Process

Before answering these questions, we study the general CPU-GPU collaborative process for a DNN training job. Fig. 4 shows the collaborative process training a DNN model on a single GPU. (1) Read data from disk into memory. (2) Pre-process raw data to a proper format for the model training. (3) Transfer data from CPU memory to GPU memory. (4) Compute gradients using GPU. (5) Update model weights. If multiple GPUs are used, global synchronization between the GPUs is required after step 4.

There are two ways to optimize the collaboration process: parallelization and pipeline. Programmers could parallelize step 1 and step 2 using the framework's interface or the self implementation. Additionally, we might refer to steps 1–3 as Stage 1. It supplies data to Stage 2, which is the major computation. Stage 1 and Stage 2 pipelines could be used to further improve performance.

Based on the above analysis, we can infer that the CPU core is important for steps 1, 2, and 5. Since these three steps run on the CPU side, they require implicit memory resources such as cache and memory bandwidth. Besides, step 3 relies on PCIe for data transfer, and multiple GPU jobs may contend for the PCIe bandwidth. Therefore, we need to analyze the DNN training job's requirements for these resources and the possible performance interference.

#### B. DNN Training Job's CPU Demand

*1) Job's Performance With CPU Core Number:* In this section, we answer the first question about the DNN training job. Since this part has been fully discussed in our conference version [36], we do not place the results and analysis due to the page limitation.

Simply put, we select state-of-the-art DL models to study how the CPU core number affects the training job's performance. Table I shows the set of representative DNN models, which target speech recognition, machine translation, question answering, speech synthesis, and image classification. We collect the DNN training job's performance under all possible CPU configurations. Besides, we also collect the job's average GPU

TABLE I  
REPRESENTATIVE DNN MODELS

Neural Model	Scenario	Type	Dataset
Alexnet [28]	CV	CNN	ImageNet
VGG16 [29]	CV	CNN	ImageNet
InceptionV3 [30]	CV	CNN	ImageNet
Resnet-50 [1]	CV	CNN	ImageNet
Bi-att-Flow (BAT) [31]	NLP	RNN	SQUAD [32]
Transformer [2]	NLP	-	WMT16 [33]
Wavenet [26]	SPEECH	CNN	VCTK [34]
DeepSpeech [3]	SPEECH	RNN	Common Voice [35]

utilization. We also use  $aNbG$  to denote the training setup of using  $a$  servers and  $b$  GPUs.

From this experiment, we can answer the first question of the DNN training job. The CPU core number affects the DNN training job's performance by impacting the CPU-side computation. **The CPU core number affects the job's performance like the traditional multithreading job.** Therefore, we can define the best-fit CPU core number of the DNN training job as the minimum CPU core number to obtain the maximal running performance. Besides, we also find that **the DNN training job also has the highest GPU utilization at the maximal training speed.**

2) *The Best-Fit CPU Number:* In this section, we try to answer the second question of the DNN training job. Specifically, we study the best-fit CPU core number for the studied models with different configurations and batch sizes (BS). Since this part have also been fully discussed in our conference version [36], we do not place the results and analysis due to the page limitation.

From the conference paper, we can answer the second question of the DNN training job. **The factors that affect the job's CPU core demand mainly include model type, data preprocessing complexity, model complexity, and pipeline optimization. Note that, CPU demands of most models are independent of batch size. This is because computation-data pipeline are both impacted by batch size.**

Although we have found these factors, we cannot directly utilize these factors for CPU demand prediction. This is because these factors cannot be quantified. We need to find other methods to automatically search the best-fit CPU number for a DNN training job (discussed Section V-B).

### C. DNN Training Job's Implicit Resource Demand

In this section, we try to answer the third question of the DNN training job. Specifically, we study DNN training job's demand for memory bandwidth, network bandwidth, and PCIe bandwidth.

1) *Memory Bandwidth:* Since this part have also been fully discussed in our conference version [36], we do not place the results and analysis due to the page limitation. From the conference paper, we could answer the question about the memory bandwidth.

We have four findings about the memory bandwidth usage. (1) When the batch size increases, its bandwidth demand increases slightly. (2) Lower-complexity CV model has a higher memory bandwidth demand. (3) Both NLP models have low

TABLE II  
THE NETWORK BANDWIDTH USAGE FOR STUDIED MODELS

Model/Network BW	Alexnet	Resnet50	Inception3	VGG16
Default BS (%)	67.1	63.4	23.2	26.9
Max BS (%)	59.4	58.8	17.6	16.2

Model	BAT	Transformer	Deepspeech	Wavenet
Default BS (%)	17.7	35.7	36.1	8.4
Max BS (%)	15.5	24.2	35.5	7.3

bandwidth requirements. (4) SPEECH models have different data pre-processing, which leads to different bandwidth usage.

We also have four findings about the memory bandwidth contention. (1) None of the models are sensitive to the solely LLC contention. (2) While the models are under the 50% bandwidth pressure, 5 of 8 models already experience a 8.1% slowdown on average. (3) The models with the 1N4G configuration are more sensible than the 1N1G configuration. (4) Different batch sizes have similar bandwidth pressure sensitivities.

#### 2) Network Bandwidth:

**Network bandwidth usage:** The DNN training jobs may also require the network bandwidth. While the training job is configured with multiple nodes, it requires the network bandwidth for gradient transmission. Table II shows the network bandwidth usage for the studied models with 2N2G configuration. All models are configured with the default batch size and max batch size. We have three observations.

First, no models except Alexnet and VGG16 have more than 40% of maximum network bandwidth usage. Second, all the models with the max batch size have a slight decline in network bandwidth usage. This is because a larger batch size brings a longer duration per iteration. Since the model's computation and the gradient's transmission are asynchronous, it has a slight decline in bandwidth usage. Third, the training job's network bandwidth usage is related to the model parameter's amount and the computation time per iteration. The greater parameter amount brings the greater bandwidth usage. The longer computation time brings the smaller network bandwidth usage.

**Network bandwidth contention:** We also conduct the experiment on the model's performance degradation under network bandwidth contention. While a CPU job uses more than 80% of the network bandwidth, all models' performance with 2N2G configuration drops by an average of 31.7%. Therefore, we also need to ensure the network bandwidth usage for DNN training jobs.

Besides, it should be noted that each node of the studied cluster is configured with two sets of network systems: a Gigabit network and an Infiniband network. The network bandwidth of the Gigabit network is 1 Gbit/s, and the network bandwidth of the Infiniband network is 10 Gbit/s. The contention experiments are conducted using the Infiniband network. DNN training jobs and CPU jobs share the Infiniband network in the experiment. Therefore, network bandwidth contention does not always exists for multi-node DNN training jobs due to two datapaths for the network.

#### 3) PCIe Bandwidth:

**PCIe bandwidth usage:** Fig. 5 shows the PCIe bandwidth usage for the studied models with the default batch size under

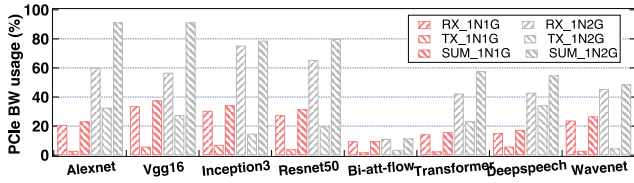


Fig. 5. The PCIe demand for different models.

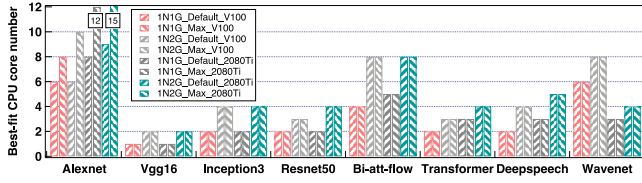


Fig. 6. The best-fit CPU core number of the studied models under different configurations under V100 and 2080Ti.

the 1N1G and 1N2G configurations. We do not consider the 1N4G case because they does not encounter the PCIe contention. We utilize *nvidia-smi* to collect each model’s read bandwidth, write bandwidth, and overall bandwidth. Since the PCIe usage data from *nvidia-smi* is coarse-grained, we average the result across multiple times.

As observed, CV models have similar PCIe bandwidth usage because of the same default batch size. SPEECH models have smaller PCIe bandwidth usage than CV models, for their batch sizes are generally less than ten. Moreover, NLP models have the smallest PCIe bandwidth usage for their data format requires little storage. While the CV models’ PCIe usage may exceed 50%, other models’ usage does not.

We do not present the experimental result with different batch sizes because *nvidia-smi* has unreasonable results. The results with the maximum batch size show that the model’s PCIe bandwidth usage exceeds the theoretical PCIe bandwidth limit. This is because the *nvidia-smi* uses a long interval to calculate the PCIe bandwidth, which means that the bandwidth data is not strictly real-time.

**PCIe bandwidth contention:** In this section, we study the performance degradation introduced by PCIe bandwidth contention. Two co-running DNN training jobs are all under 1N2G configuration. We bind the two jobs to different memory nodes to isolate the impact of memory bandwidth.

Experimental results show that the noticeable performance drop exists only when one of the co-located training jobs is Alexnet or Resnet50. The performance degradation is between 5%–10% (except Transformer co-running with CV models have 30%). Therefore, co-running DNN training jobs may encounter the PCIe contention. Despite this, it is hard to model the performance degradation because of limited and inaccurate information.

#### D. Generality on Other GPU Types

Fig. 6 shows the best-fit CPU core number of the studied models with different configurations under V100 and 2080Ti. From this figure, we have two conclusions. (1) The best-fit CPU core number of the same model on different GPUs is different.

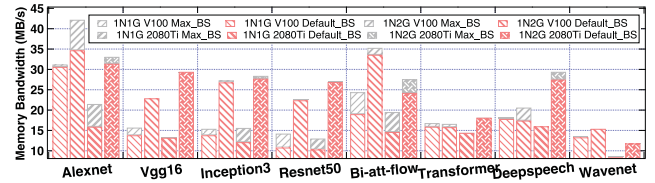


Fig. 7. The memory bandwidth usage for the studied models with different configurations under V100 and 2080Ti.

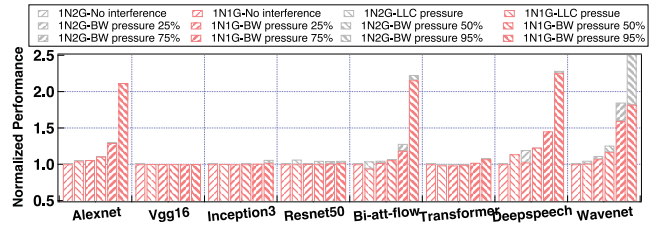


Fig. 8. The normalized performance of all the 1N1G/1N2G models under contention with V100.

The factors that affect the job’s CPU core demand include the GPU type, the model structure, the CPU core type. (2) The way that CPU core affects the performance is not related to the GPU type. The CPU core number affects the job’s performance on all GPU types like the traditional multithreading job.

Fig. 7 shows the memory bandwidth usage for the studied models with different configurations under V100 and 2080Ti. From this figure, we can infer the same conclusions in Section IV.C.1. First, different models have different bandwidth requirements. Second, when the batch size increases, its memory bandwidth increases slightly.

Fig. 8 shows the performance degradation under memory contention with V100. The experimental results with 2080Ti are similar, which are not shown due to page limitation. From this figure, we can also infer the same conclusions as Section IV.C.1. First, no models suffer from the LLC contention. Second, while the models are under the 50% bandwidth pressure, 5 of 8 models already experience the performance degradation. Third, the models with 1N2G configuration are more sensible than 1N1G configuration.

Based on the above experiments, we can get the conclusion that all the insights obtained in this paper are common to other GPU types.

#### E. DNN Inference Job’s CPU Usage

1) *DNN Inference Job’s Questions:* The GPU cluster may host multiple user-facing services relying on DNN inference jobs. The scheduling system on the cluster launches or terminates inference jobs based on each service’s query load. User queries need the get the inference result within the quality-of-service target (QoS target), which is generally set as 200 ms. The studied cluster provides 12 services with DNN models, and they all rely on Tensorflow-serving [37]. **Therefore, these jobs could be scheduled using CPU or GPU without modification.**



TABLE III  
THE LARGEST BS FOR DIFFERENT MODELS UNDER 16 CPU CORES

Model	Alexnet	Resnet50	Inception3	VGG16
Largest BS	16	4	2	1
Model	BAT	Transformer	Deepspeech	Wavenet
Largest BS	1	8	0	0

TABLE IV  
BANDWIDTH USAGE FOR DIFFERENT MODELS WITH THE LARGEST BATCH SIZE

Model	Alexnet	Resnet50	Inception3	VGG16
BW (GB/s)	25.7	26.9	27.0	40.3
Model	BAT	Transformer	Deepspeech	Wavenet
BW (GB/s)	29.5	26.3	N/A	N/A

Since GPU jobs also require CPU cores, DNN inference jobs cannot use all the CPU resources. Therefore, the first question about DNN inference jobs is whether the inference job could satisfy the user query's QoS target with limited CPU resources. Based on the analysis in Section IV-B, we choose 12 CPU cores to represent the CPU cores occupied by GPU jobs. Since our experimental machine has 28 or 56 CPU cores, we study whether the inference job could utilize the remaining 16 CPU cores to serve the user query.

Table III shows the largest batch size that the inference job could provide the service within the QoS target. Six models could utilize 16 CPU cores to host the DNN inference job, while the remaining two models cannot. For most models, the DNN inference job only involves the forward computation so that the CPU cores could satisfy the demand.

Table IV shows the bandwidth usage for all models with the batch size in Table III. As observed, none of the models use more than 60% of the theoretical bandwidth (70GB/s), mostly about 40%. We also conduct the experiment to co-run DNN training jobs and DNN inference jobs. Experimental results show that neither jobs encounter performance degradation due to memory bandwidth contention.

Therefore, we could answer the DNN inference job's two questions. When some DNN inference jobs could utilize limited CPU cores to provide the service within the QoS target, they could utilize the CPU cores on the GPU node for computation. Moreover, there is no performance interference between DNN training jobs and DNN inference jobs for studied models.

2) *DNN Inference Job's Performance With CPU Number:* Since the DNN inference job may support a large batch size, the optimal configuration for a DNN model's inference job is unknown. Fig. 9 shows the maximum batch size that three models could support and its bandwidth usage under different CPU cores.

From this figure, we have two observations. First, the largest batch size that one model's inference job could support increases with the CPU core number. For the model with high complexity, the increase in CPU core does not always lead to the increase in the batch size. Second, the bandwidth required for the DNN inference job also has a linear relationship with the CPU core number. The slope of the bandwidth usage is different from that of the batch size.

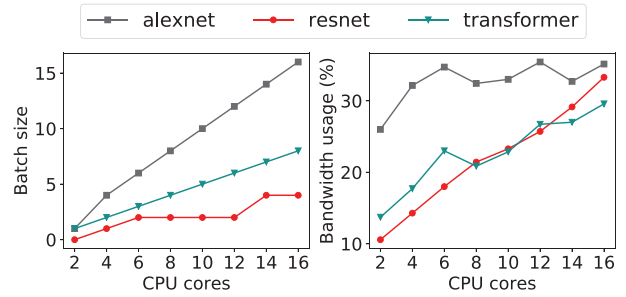


Fig. 9. The batch size and bandwidth usage for different inference jobs.

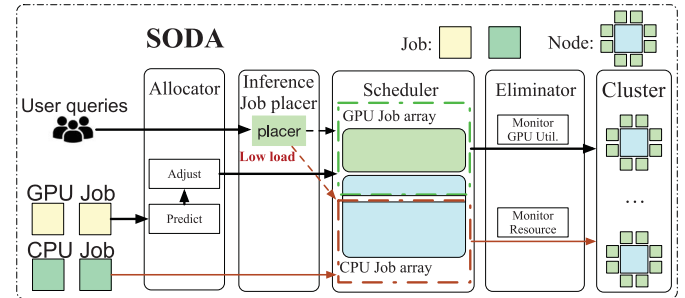


Fig. 10. Design overview of SODA.

## V. METHODOLOGY OF SODA

In this section, we present SODA that solves the GPU cluster's four problems. SODA is a job scheduling system that improves cluster utilization by adaptive resource management. SODA targets the multi-tenant GPU cluster, which co-locates DNN jobs and CPU jobs.

### A. Overview

Fig. 10 shows the design overview of SODA. SODA consists of an *adaptive CPU allocator*, a *multi-array job scheduler*, a *hardware-aware inference job placer*, and a *real-time contention eliminator*. These four modules solve the GPU cluster's four problems, respectively.

Specifically, the CPU allocator determines the best-fit CPU core number for a DNN training job. The multi-array job scheduler manages the resource division and the job scheduling. The inference job placer determines the computing resource used by the inference job based on the runtime query load. The contention eliminator ensures that the performance of DNN training jobs would not be adversely affected by CPU-side resource contention. Apart from the above four parts, SODA periodically updates the job information from all users and array-level job information in the backend. The information is helpful for the CPU allocator and job scheduler to make efficient decisions.

As for DNN training jobs and CPU jobs, SODA schedules a newly received job  $J$  as follows. 1) If  $J$  is a DNN training job, the CPU allocator searches the best CPU number for  $J$ . 2) If  $J$  is a DNN training job,  $J$  is pushed into the GPU job array. If not,  $J$  is pushed into the CPU job array. 3) The job scheduler assigns  $J$  the required resources based on the status of all the nodes.

4) The contention eliminator on each node keeps monitoring each job's memory and network bandwidth usage, and throttles the bandwidth of a CPU job if it consumes excessive bandwidth. 5) When  $J$  completes, its resource usage, scheduling information, and owner information are recorded in a log for future use.

As for DNN inference jobs, SODA determines the inference job's launch and termination based on the query load [7], [8]. While the optimal CPU configuration for one model's inference job is profiled offline, SODA determines the inference job's resource type at runtime.

### B. Adaptive CPU Allocation

For a DNN training job  $J$ , the adaptive CPU allocator determines its best-fit CPU core number. Though the factors impacting the training job's CPU demand cannot be quantified, we still have two observations in Section III. First, a DNN training job's GPU utilization and running speed change in a same trend, and reach the optimal value at the same CPU number. Second, the CPU core number affects the job's performance like traditional multithreading jobs. If the CPU core number exceeds the best-fit number, the training job's GPU utilization does not change or drop slightly.

We design a feedback-based adaptive CPU allocator based on these two observations. The allocator increases or decreases the CPU core number allocated to  $J$  to check whether more or fewer cores would result in a higher performance. The CPU allocator searches the best-fit CPU core number in two steps. First, the allocator finds a reasonable CPU core number as the start point. Second, the allocator adjusts the CPU core number to locate the best-fit CPU core number for optimal performance.

1) *Determining  $N_{start}$* : According to the discussion in Section IV-A, models belonging to the same category have similar pre-processing. We presume that the tenant provided at least the model's category, and optionally the following three types of information: the model weight's amount, whether to use pipeline optimization, and data processing complexity between iterations.

In general, a user tends to submit similar training jobs. Based on this assumption,  $N_{start}$  for job  $J$  is determined based on the numbers of cores allocated to its owner's previous job in the same category of  $J$ . In more detail, we choose the largest core number to be  $N_{start}$ . If  $J$  is the first job submitted by a tenant, we choose 3 for CV models, 5 for NLP models, and 5 for SPEECH models based on our investigation in Section IV-B. In the worst case that the owner of  $J$  does not even offer the category of  $J$ , it is also sufficient to find a reasonable  $N_{start}$  based only on the owner's historical job information.

If the owner of  $J$  provides further information, the  $N_{start}$  can be optimized further in light of the findings in Section IV-B. To be more specific, if  $J$  is implemented with pipeline optimization, the relevant  $N_{start}$  is reduced by 1. If  $J$  has a large number of job weights,  $N_{start}$  is reduced by 1. If the processing complexity between iterations of  $J$  is high,  $N_{start}$  is increased by 1.

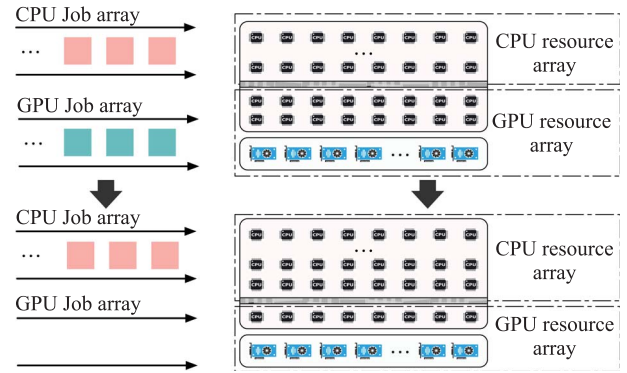


Fig. 11. The design of the multi-array job scheduling.

2) *Tuning the Core Number*: Beginning with  $N_{start}$ , SODA tries both larger and smaller core number allocated to  $J$  (denoted by  $N_{opt}$ ), and checks whether they improve  $J$ 's performance. The CPU allocator first evaluates the smaller core number for  $J$ . Three scenarios are possible here. 1) If GPU utilization increases, the allocator decreases  $N_{opt}$  until GPU utilization remains unchanged. 2) If the smaller core number does not improve the GPU utilization, the allocator increases  $N_{opt}$  in the other direction. If increasing  $N_{opt}$  improves the GPU utilization, the allocator keeps increasing  $N_{opt}$  until GPU utilization remains unchanged. 3) If neither less nor more cores improve GPU utilization, the best-fit number of cores for  $J$  is discovered. In Section VI-H, we analyze the tuning's effectiveness and associated overhead.

3) *Generality*: Our adaptive CPU allocation method stems from the iteration-based computation mode of the DNN training job. If a job also has this iterative-based mode, such as fluid dynamics computation in the scientific computing, it could also use the adaptive CPU allocation method for the best-fit CPU core number. On the contrary, if a job does not have iterative computations, such as one-shot computation, then this method is not suitable.

### C. Multi-Array Job Scheduling

Fig. 11 illustrates our multi-array approach. On the one hand, we divide the cluster's job array into the CPU job array and the GPU job array. The GPU job array is further subdivided into the 1-GPU job array and the 4-GPU job array. On the other hand, we divide the CPU resources on the cluster into the CPU job's CPU resource array and the GPU job's CPU resource array. The GPU resources on the cluster are divided into the 1-GPU resource array and the 4-GPU resource array. The division of the computing resources is derived from historical statistical information.

When one job arrives, it is distributed to different job arrays based on the resource requirement. In most cases, the job only use the resource in the associated resource array. Besides, if CPU jobs burst and the GPU job's CPU resource array is relatively idle, the multi-array scheduler enables CPU jobs to preempt the reserved cores in the GPU job's CPU resource array (Fig. 11). When a GPU job arrives and requires the preempted CPU cores, SODA aborts the CPU job and releases the CPU



cores. The suspended CPU job re-enters the array head, waiting to be rescheduled again. Benefit from containerization and virtualization (Docker and Kata), job migration is easy to achieve here.

As for the GPU jobs arrays, the 4-GPU sub-array is for jobs that apply for 4 GPUs and more, while the 1-GPU array serves jobs that require less than 4 GPUs. Similarly, when the resources in the 4-GPU array are all used, the GPU job in the array will be allocated to the nodes that fulfill the requirements in the 1-GPU array. If no suitable node is found, the job queues for scheduling later. Similarly, if all resources in the 1-GPU array are consumed, the job tries to preempt resources from the 4-GPU job array. When 4-GPU jobs need to use the resources, job migration is performed.

If there are multiple GPUs in the cluster, we create their own GPU array for each of them. This is because the user will specify the GPU type they need. The scheduler just need to make the scheduling decision based on the GPU type requested by the user. For each GPU array, SODA uses DRF for job scheduling.

The multi-array design will result in a limited CPU core number for CPU jobs. However, this has limited impact on CPU jobs. This is because most CPU jobs themselves are distributed, such as data processing tasks. If a distributed CPU job applies for excessive CPU cores for one node, we could directly adjust it to the same number with different configurations. Meantime, if a single-node CPU job applies for excessive CPU cores, we allocate the maximum CPU core number for it and inform them of the CPU core number available on one node later. In this case, the multi-array design will not cause CPU core fragmentation.

#### D. Hardware-Aware Inference Job Placement

1) *Original Service Mode*: The GPU cluster may host multiple user-facing services using DNN models. For example, the studied cluster provides 12 services. Each service maintains a service manager, which forwards user queries to corresponding inference jobs. Meanwhile, the service manager launches or terminates inference jobs according to the real-time query load.

Specifically, each inference job is a long-running serving system. The serving system is configured with a time window  $time_{window}$  and a max batch size  $batch_{max}$  [38]. Once the serving system collects  $batch_{max}$  user queries within the time window, it starts computing immediately. When not enough queries are collected within the time window, it starts performing computation with the current batch size. The  $time_{window}$  and  $batch_{max}$  are set based on the QoS target, and the overall duration should be less than the QoS target. Generally, the computation stage and query collection stage are pipelined.

Since each inference job could serve at most  $batch_{max}$  queries each time, the service manager can calculate the maximum load  $load_{max}$  that the service can support based on the number of inference jobs. If the query load for a period of time continues to exceed a threshold (for example, 80% of  $load_{max}$ ), the service manager launches a new inference job to increase the  $load_{max}$  of the service. If the query load exceeds the maximum load, the service manager directly launches new inference jobs

to meet their needs. The above service management method is consistent with the service management method on AWS Lambda [39].

However, the user queries' diurnal pattern means few queries at night [13]. Since the inference jobs must reside on the GPU for possible coming queries, the GPU occupations at night have low GPU utilization. These low-utilized GPUs could not serve DNN training jobs with high utilization.

2) *Inference Job Placement*: We have three insights in Section IV-E. (1) The inference jobs mostly rely on deep learning frameworks, such as Tensorflow serving [37]. They can be scheduled using CPU or GPU with no modification. (2) One inference job could use CPU cores for computation only when two conditions are met. It could utilize limited CPU cores to provide the service within the QoS target, and it does not encounter memory bandwidth contention. (3) The inference job's best configuration is the smallest CPU core number that supports the largest batch size. Based on these three insights, we design a hardware-aware inference job placement method, which is divided into online and offline stages. All the inference jobs serve the queries in the same mode, which all support dynamic batch size.

In the offline stage, we first use the largest CPU core number to profile the largest batch size that the inference job could support. The largest CPU core number comes from the CPU job's CPU resource array mentioned in the previous section. If the batch size is smaller than 1, this service could not use CPU cores for the inference process. On the contrary, we find the smallest core number that supports the batch size and its memory bandwidth usage. Besides, we still profile the maximum batch size that one GPU could support, just like the traditional works.

In the online stage, we choose the largest batch size  $bs_{gpu}$  that one GPU could support as the switching point. When the query load is less than the  $bs_{gpu}$ , the placer transforms the inference job from one GPU job to multiple CPU jobs. As for the CPU job scheduling, we monitor each node's memory bandwidth usage to avoid the contention between training jobs and inference jobs. When the query load is greater than  $bs_{gpu}$ , the placer submits one GPU job to replace these CPU jobs. Besides, when many CPU jobs are queuing, and the GPU resource array is idle, we still choose GPU for the inference job. When some GPU jobs are queuing and the query load is high, we set the inference job as the first-priority job among all the jobs.

Furthermore, we enable the inference job to preempt the resources from the DNN training jobs. If there are not enough resources when the inference job is submitted. SODA enables the inference job to preempt the resources from DNN training jobs and CPU jobs. SODA chooses the last submitted job and puts them at the head of the job queue. This is because jobs submitted earlier are more likely to complete the computation, which could alleviate the resource crisis in the cluster.

3) *Discussions*: Note that, many previous works rely on offline analysis to build performance models. For example, several works [38], [40], [41] require great effort on offline analysis. The work from KAIST [40] need to profile total 1,250

job pairs, and Abacus [38] needs 2 hour profiling for each job-pair profiling. For SODA, the job placer only needs to profile the optimal CPU configuration for each model, which only needs 5 minutes at most. Therefore, the profiling process of SODA is simple and effective.

The co-location between multiple DNN inference jobs could also reduce unnecessary GPU usage to some extent. However, the co-location of DNN inference jobs still needs to occupy a certain number of GPUs. This is because the co-location is limited by the GPU memory and query load. For example, Abacus only co-locates three to four services on one GPU [38]. Since SODA could utilize CPU resources to serve DNN inference jobs, it could better reduce unnecessary GPU usage.

We do not focus on the co-location between DNN inference job and DNN training job in this paper. This is because, this will seriously affect the performance of DNN training job. Experimental results show that a DNN training job under co-location suffers up to  $11\times$  performance degradation compared to normal queuing for execution. Since another important goal of this paper is the short queuing and better performance of DNN training job, we do not focus on the co-location of DNN inference jobs and DNN training job.

If there are multiple GPUs on the cluster, we first select the GPU with the least usage. This is because the DNN inference job will affect the scheduling of other GPU jobs. If the DNN inference job always uses a specific GPU type, other training jobs using this GPU type will encounter long queuing time. Therefore, we choose the most idle GPU array to undertake the DNN inference job.

### E. Real-Time Contention Elimination

Concerning memory bandwidth contention, SODA monitors the total memory bandwidth usage of each node and the memory bandwidth of each CPU job on the node using Intel Memory Bandwidth Monitoring (MBM) technique. If the total memory bandwidth usage of the node reaches a pre-defined threshold (75% by default according to the analysis in Section IV.C.1), the performance of the DNN training jobs degrades due to the memory bandwidth contention.

In this scenario, the contention eliminator throttles the memory bandwidth of CPU workloads via the Memory Bandwidth Allocation (MBA) approach. If the node does not support the MBA technique that only works on the latest CPU, the contention eliminator reduces the number of cores allocated to the CPU jobs by half. This method reduces the memory bandwidth usage of the CPU job and eliminates the performance degradation of the DNN training job due to the memory bandwidth contention. For the released CPU cores, SODA tries to schedule new CPU jobs and uses the same method to ensure that the GPU utilization is not affected.

If the node does not support memory bandwidth monitor, SODA can detect memory bandwidth contention by monitoring the GPU utilization of a DNN training job. During offline analysis, SODA records the stable GPU utilization of the training job. After the job is scheduled on the cluster, SODA monitors its GPU utilization. If the GPU utilization drops significantly,

SODA adjust the memory bandwidth usage of CPU jobs on the same node.

As for the network bandwidth contention, SODA first filters the nodes hosting the GPU jobs with the multi-node configuration because these tasks may suffer from network contention. When there are GPU jobs with multi-node configuration on one node, SODA uses the Linux command *tc* [42] to control the network bandwidth usage. Specifically, we configure 75% of the network bandwidth for GPU jobs, while CPU jobs can only use 25% (Section IV.C.2). These two values could be adjusted with one cluster's specific network bandwidth. If there are two network systems, we only configure GPU jobs to use the high-bandwidth network and CPU jobs to use the low-bandwidth one.

As for PCIe bandwidth contention, SODA does not perform the management for two reasons. First, all the GPU jobs have the same priority. Second, there are no relative PCIe-related tools to control PCIe usage.

Note that, although DNN training jobs suffer from the bandwidth contention from CPU jobs, it is hard to locate the performance degradation without the profiling. Based on that, profiling all the DNN jobs and CPU jobs before scheduling incurs great overhead. Therefore, we do not consider the DNN training job's scheduling based on the bandwidth usage, and only guarantee their performance by eliminating the contention from CPU jobs.

## VI. EVALUATION OF SODA

### A. Experimental Setup

We collected the task trajectory information for one month from the real-production cluster. There are 105,000 jobs are submitted during one month, of which 75,000 jobs are CPU jobs, and 30,000 jobs are GPU jobs. Most of the GPU jobs are related to the NLP and SPEECH models. 25,000 of the GPU jobs are DNN training jobs, and the rest are DNN inference jobs. Moreover, there are also a large number of CPU jobs for auxiliary tasks and other tasks. The specific experimental configuration has been explained in Section III, which has 80 nodes and 400 NVIDIA 1080Ti GPUs.

Based on the real-system traces, we compare SODA with three systems: KELP-FIFO (KELP [10] with First In First Out), KELP-DRF (KELP with Dominant Resource Fairness), and CODA (The conference version [36]). KELP improves the cluster's resource utilization by eliminating the memory bandwidth contention between CPU jobs and DNN jobs. KELP-FIFO schedules the DNN jobs and CPU jobs in the FIFO manner. KELP-DRF considers GPU the dominant resource and enforces the fair share of the dominant resource.

### B. Improving Resource Utilization

Figs. 12 and 13 show the GPU active rate and the GPU utilization with KELP-FIFO, KELP-DRF, CODA, and SODA. As observed from Fig. 12, SODA significantly improves the GPU utilization compared with KELP-FIFO, KELP-DRF, and CODA. The GPU utilization of the cluster with KELP-FIFO,

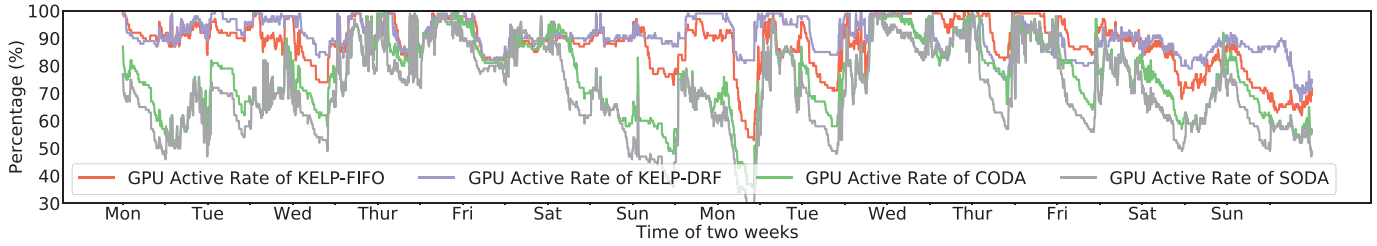


Fig. 12. The GPU active rate of the multi-tenant GPU cluster with KELP-FIFO, KELP-DRF, CODA, and SODA.

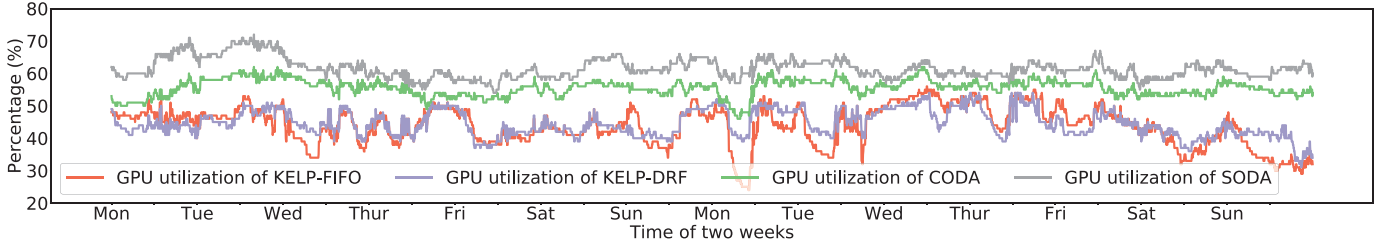


Fig. 13. The GPU utilization of the multi-tenant GPU cluster with KELP-FIFO, KELP-DRF, CODA, and SODA.

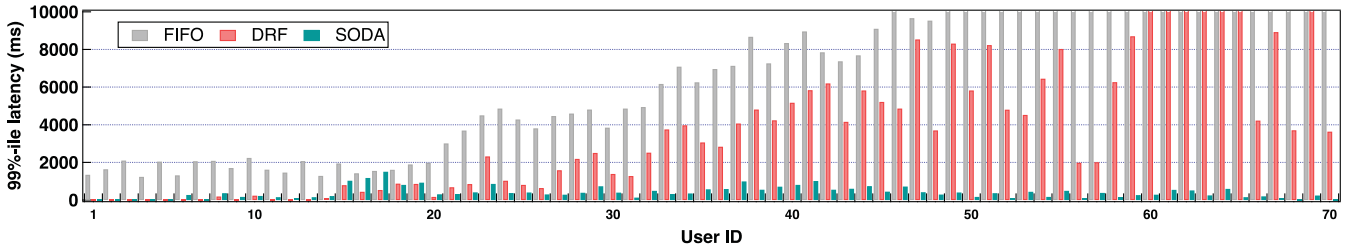
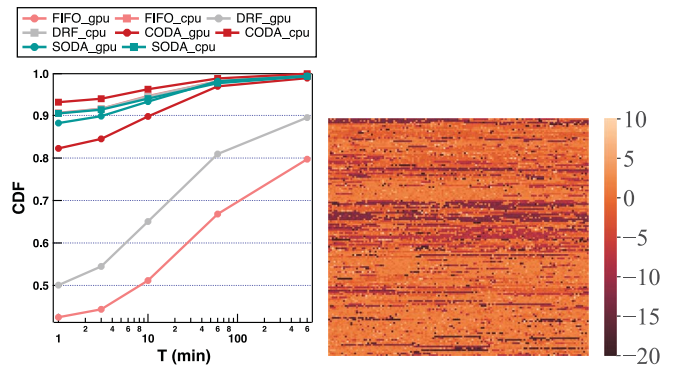


Fig. 14. Each user’s 99%-ile queuing time with SODA, FIFO, and DRF.

KELP-DRF, CODA, and SODA is 45.5%, 45.1%, 59.8%, and 65.4%, respectively. SODA improves the GPU utilization of the GPU cluster by  $65.4\% - 45.5\% = 19.9\%$ . At the same time, The GPU active rate of the cluster with KELP-FIFO, KELP-DRF, CODA, and SODA is 90.2%, 89.7%, 80.3%, and 76.3%. SODA has the lowest GPU active rate because it could improve the performance of DNN training jobs and avoid unnecessary GPU occupation of DNN inference jobs. The low GPU active rate implies that SODA could serve higher job throughput.

The improved GPU utilization originates from the adaptive CPU allocation, the hardware-aware inference job placement, and the real-time contention elimination in SODA. The adaptive CPU allocator in SODA selects the best-fit CPU number for each job. It not only avoids the unreasonable CPU requirement and its induced fragmentation but also optimizes the performance of the DNN training jobs. The hardware-aware inference job placer reduces the unnecessary GPU occupation of the inference jobs and mitigates the GPU resource contention. The real-time contention eliminator exempts the GPU jobs from memory and network contention, which further optimizes the GPU resource usage.

By comparing the GPU active rate curve of SODA with the GPU active rate curve of KELP-FIFO in Fig. 12, we can find that SODA reacts to the job change in advance. This is because SODA reduces the job queuing time by optimizing job scheduling and resource allocation. Specifically, SODA reduces



(a) CDF of job queuing time. (b) CPU/GPU ratio of GPU jobs.

Fig. 15. Information of DNN training jobs.

the GPU fragmentation through multi-array job scheduling and adaptive CPU allocation. Besides, SODA avoids the unnecessary GPU occupation and mitigates the memory and network contention encountered by the GPU jobs. These experimental results are shown in later sections.

C. Reducing the Queuing Time

Figs. 15(a) and 14 show the CDF of all the jobs’ job queuing time and the 99%-ile job queuing time of each user with KELP-FIFO, KELP-DRF, CODA, and SODA. With KELP-FIFO and



KELP-DRF, 57.5% and 49.5% of GPU jobs suffer from queuing time of more than 1 minute, 33.2% and 19.1% of GPU jobs queue up for more than 1 hour. Besides, 90.5% and 90.6% of the CPU jobs can get resource allocation within 10 seconds. With CODA and SODA, 82.2% and 88.2% of GPU jobs can get resource allocation without queuing, and 93.9% and 91.3% of CPU jobs can be scheduled to the cluster within 1 minute.

SODA's short queuing time is due to its optimized CPU allocation, multi-array scheduling, and inference job placement. First, the CPU allocation optimization addresses the problem of excessive CPU application from GPU jobs, hence reducing GPU fragmentation. Second, SODA mitigates the impact of bursty CPU jobs through its multi-array design. Third, hardware-aware inference job placement reduces the unnecessary GPU occupation, which mitigates the GPU resource contention.

As observed from Fig. 14, the queuing time for most users is longer with KELP-FIFO than with KELP-DRF. KELP-DRF provides better fairness, users who submit a large number of jobs have a longer queuing time, waiting for tasks of users applied for fewer resources. SODA has a significantly shorter queue time for all users than KELP-FIFO and KELP-DRF. As previously stated, SODA reduces GPU fragmentation and increases job performance, which improves system throughput and optimizing job queuing time. Besides, SODA also ensures fairness among users since the DRF algorithm is used for scheduling inside each array. As the figure indicates, users that submit more tasks experience a longer queuing time.

#### D. Effectiveness of Tuning CPU Allocation

Fig. 15(b) presents the tuning of the core number allocated to each DNN training job with SODA. As shown in the figure, 57.1% of the GPU jobs are allocated 1–5 more cores, and 33.6% of the GPU jobs are allocated 1–20 fewer cores compare with the number of cores applied by the job owner. This is consistent with the investigation in Section III: many DNN training jobs apply for one or two cores for each GPU, and a considerable percentage of the DNN training jobs request excessive CPU cores.

Fig. 19 shows the end-to-end latency of representative GPU jobs (including queuing time and processing time). The left and right bars reflect the queuing and processing times for each job using FIFO and SODA, respectively. As illustrated in the figure, SODA concurrently reduces the queuing and processing times of the majority of jobs. The queuing time reduction comes from the improvement of the overall cluster throughput, while the processing time reduction is due to the adaptive CPU allocator's CPU adjustment.

#### E. QoS and Co-Location Experiments

Fig. 16 shows the 99%-ile latency of the baseline serving method and the hardware-aware inference job placer. In this experiment, the QoS target is set as 200ms. We configure each service with the batch size that have the computation time closing to 100ms, and set the time window same as the computation time. For example, the Resnet50 using CPU under batch size 2

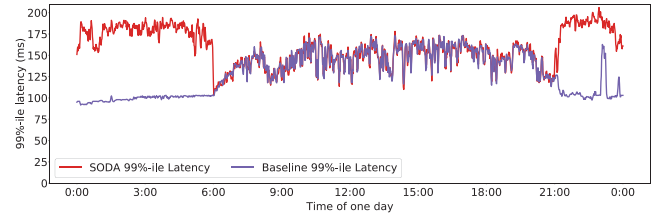


Fig. 16. The 99%-ile latency of queries under baseline method and SODA.

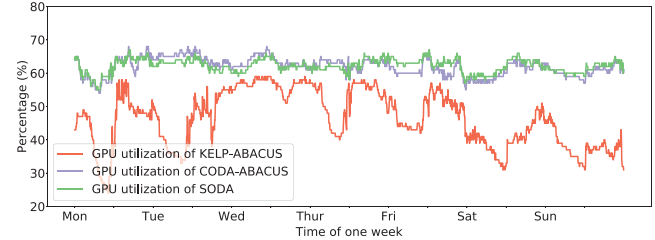


Fig. 17. The GPU utilization of the multi-tenant GPU cluster with KELP-ABACUS, CODA-ABACUS and SODA.

needs 95.96ms to finish the computation, and the time window is also set as 95.96ms.

As observed from the figure, the baseline serving method does not cause any QoS violations at first. This is because it configures redundant resources in advance to guarantee the query's latency. Second, under the high load during the daytime, the inference job placer has similar 99%-ile latency with baseline method. Since the inference placer also rely on GPU for query serving under high load, it has the similar 99%-ile latency with the baseline method. Third, under the low load at nighttime, the inference job placer brings higher 99%-ile latency. While GPU could provide better computing power than CPU, the inference jobs using GPU could get better latency performance. Despite this, the job placer does not introduce any QoS violations.

Fig. 17 presents the GPU utilization of KELP-ABACUS, CODA-ABACUS and SODA. The first two systems are enhanced by the DNN inference jobs co-location from ABACUS [38]. The GPU utilization of the cluster with KELP-ABACUS, CODA-ABACUS, and SODA is 48.4%, 64.5%, and 65.4%, respectively. Compared with CODA-ABACUS, SODA still has better performance. This is because the co-location of DNN inference jobs could only partially reduce the unnecessary GPU occupation. Furthermore, if there are more services on the cluster, SODA will enjoy greater performance advantages.

In addition, the co-location of DNN inference jobs requires great effort on offline analysis, which may take tens of hours. During this period, the job co-location could not solve the unnecessary GPU usage. On the contrary, SODA only needs a few minutes to complete the profiling of DNN inference job. It supports the resource switching of new services seamlessly.

#### F. Effectiveness of Placing Inference Jobs

Fig. 18 shows the CPU active rate with KELP-FIFO, KELP-DRF, CODA, and SODA. As observed from Fig. 18, SODA

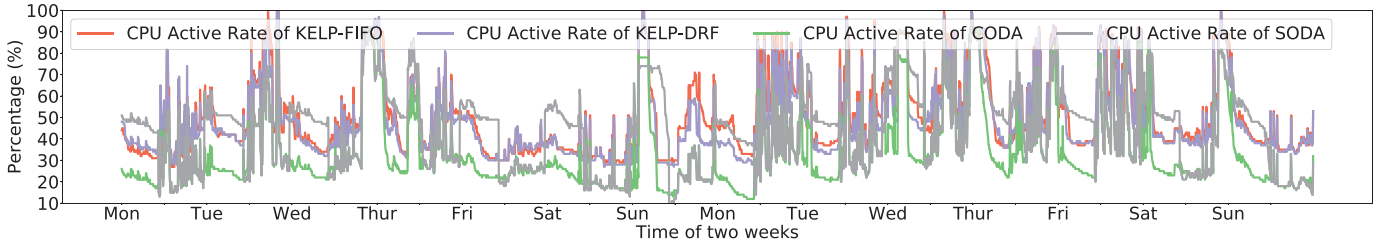


Fig. 18. The CPU active rate of the multi-tenant GPU cluster with KELP-FIFO, KELP-DRF, CODA, and SODA.

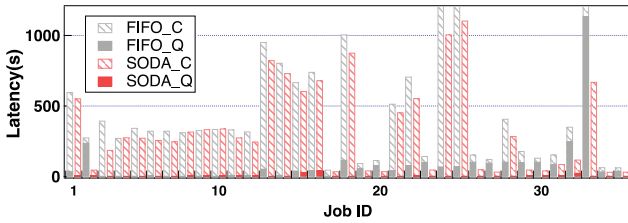


Fig. 19. The end-to-end latencies of representative GPU jobs with FIFO and SODA.

improves the CPU active rate compared with CODA, and it has a lower CPU active rate compared with KELP-FIFO and KELP-DRF. The CPU active rate of the cluster with KELP-FIFO, KELP-DRF, CODA, and SODA is 52.8%, 54.5%, 35.1%, and 46.4%, respectively.

The lower CPU active rate compared with KELP-FIFO and KELP-DRF comes from the optimized CPU allocation of the DNN training jobs. Although SODA has a lower CPU active rate, it brings more efficient resource usage. The improved CPU active rate compared with CODA originates from the hardware-aware inference job placer, which also brings the improved GPU utilization. Observed from the figure, the GPU active rate of SODA drops in the nighttime while the CPU active rate of SODA increases simultaneously. This demonstrates the effectiveness of inference job placement. It perceives the query load difference and chooses the best-fit hardware for the inference jobs.

### G. Effectiveness of Eliminating CPU-Side Contention

To determine the effectiveness of SODA's contention eliminator, we disable it and retest performance. Our experimental results indicate that when jobs are queued, the average GPU utilization decreases by 2.2% if the contention eliminator is disabled. Meanwhile, the queuing jobs double.

The above performance data is reported in the scenario that only 0.5% of CPU tasks have high memory or network bandwidth requirements. If more CPU jobs on the cluster have higher memory bandwidth requirements, the performance is worse without the contention eliminator. Since KELP only focus on the memory bandwidth management, it could not handle the network bandwidth contention. Besides, SODA utilize simple memory bandwidth interfaces and core throttle to manage the memory bandwidth contention, and KELP adopts complicated memory bandwidth division, which brings heavy overhead.

TABLE V  
OVERHEAD OF IDENTIFYING THE OPTIMAL CORE NUMBER

Neural Model	Profiling Steps	Training Iterations
Alexnet [28]	4	about 170
VGG16 [29]	4	about 47
InceptionV3 [30]	3	about 120
Resnet-50 [1]	3	about 100
Bi-att-Flow [31]	4	about 23
Transformer [2]	3	about 170
Wavenet [26]	3	about 20
DeepSpeech [3]	3	about 30

### H. Overhead of Identifying the Core Number

When determining the optimal core number for a DNN training job, SODA profiles the job's performance using different core numbers. Throughout this process, we sample the GPU utilization for each profiling step that lasts 60 seconds. As illustrated in Table V, SODA determines the best-fit core number for all DNN training jobs in four profiling steps, which needs four minutes. The table also lists the number of iterations that each model has been trained during the profiling step. Each model is trained for between 20 and 170 iterations, which is sufficient to determine the CPU requirements of jobs. Additionally, we analyze the runtime distribution of GPU jobs in a week and discover that 39.6% of DNN training jobs last longer than 2 hours and 68.5% last longer than an hour. It is worth spending about four minutes to explore the best-fit core number.

The allocator increases or decreases the CPU core number to check whether more or fewer cores would result in a higher performance. However, the job launch and context preparation of the DNN training job also take a long time, which may take more than 3 minutes. Faced with this problem, we set a large parallelism at the beginning. After sampling for a period of time, we use *taskset* to dynamically adjust the parallelism on the CPU side. In this way, we can avoid the overhead caused by the job launch.

### I. Generality

Some larger private clusters may have both GPU and CPU nodes. FIFO scheduling still results in low GPU utilization and GPU fragmentation for these clusters. In addition to existing problems, DRF is confronted with additional ones. When GPU resources are more scarce than CPU resources, a tenant who submits both CPU and GPU jobs can quickly accumulate a significant weight. Then its CPU jobs would no longer be scheduled. This situation conveys unfairness among users. However,

SODA's multi-array scheduling ensures that GPU and CPU jobs are not affected by each other.

## VII. CONCLUSION

A multi-tenant GPU cluster hosts both DNN training jobs and traditional CPU jobs. We characterize the CPU-side resource demand and contention of training DNN models in Speech, CV, and NLP field. Besides, we explore the possibility to utilize the CPU cores on GPU node for DNN inference jobs. Based on the analysis, we propose SODA, a scheduling system that improves the resource utilization of GPU clusters. SODA could find the just-enough core for DNN training jobs and exempt the CPU-side resource contention. Meanwhile, SODA could determine the best-fit resources for DNN inference jobs. Experimental results show that SODA improves the GPU utilization by more than 19.9%, while all the DNN inference jobs could provide the service within the quality-of-service target and the queuing performance of CPU jobs does not get degradation.

## REFERENCES

- [1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.
- [2] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1–11.
- [3] A. Hannun et al., "Deep speech: Scaling up end-to-end speech recognition," 2014, *arXiv:1412.5567*.
- [4] Y. Jia et al., "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*. New York, NY, USA: ACM, 2014, pp. 675–678.
- [5] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in *Proc. OSDI*, 2016, pp. 265–283.
- [6] K. Hazelwood et al., "Applied machine learning at Facebook: A data-center infrastructure perspective," in *Proc. IEEE Int. Sym. High Perform. Comput. Architect. (HPCA)*. Piscataway, NJ, USA: IEEE, 2018, pp. 620–629.
- [7] H. Shen et al., "Nexus: A GPU cluster engine for accelerating DNN-based video analysis," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 322–337.
- [8] W. Cui, Q. Chen, H. Zhao, M. Wei, X. Tang, and M. Guo, "E2bird: Enhanced elastic batch for improving responsiveness and throughput of deep learning services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1307–1321, Jun. 2020.
- [9] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," 2019, *arXiv:1901.05758*.
- [10] H. Zhu, D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and M. Erez, "Kelp: QoS for accelerated machine learning systems," in *Proc. IEEE Int. Sym. High Perform. Comput. Architect. (HPCA)*. Piscataway, NJ, USA: IEEE, 2019, pp. 172–184.
- [11] "First in first out algorithm." 2022. [Online]. Available: [https://en.wikipedia.org/wiki/FIFO\\_\(computing\\_and\\_electronics\)](https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics))
- [12] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. NSDI*, vol. 11, no. 2011, 2011, pp. 24–24.
- [13] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: QoS awareness and increased utilization for non-preemptive accelerators in warehouse scale computers," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 681–696, 2016.
- [14] V. K. Vavilapalli et al., "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. SOCC*, 2013, pp. 1–16.
- [15] B. Hindman et al., "Mesos: Flexible resource sharing for the cloud," *USENIX Mag.* vol. 1, no. 1, pp. 1–12, 2011.
- [16] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *Proc. EuroSys*. New York, NY, USA: ACM, 2013, pp. 365–378.
- [17] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. EuroSys*, 2010, pp. 265–278.
- [18] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. EuroSys*. New York, NY, USA: ACM, 2018, Art. no. 3.
- [19] Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, and L. Tang, "Prophet: Precise QoS prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers," *ACM SIGOPS Operating Syst. Rev.*, vol. 51, no. 2, pp. 17–32, 2017.
- [20] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "AlloX: Compute allocation in hybrid clusters," in *Proc. EuroSys*, 2020, pp. 1–16.
- [21] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking beyond {GPUs} for {DNN} scheduling on {multi-tenant} clusters," in *Proc. OSDI*, 2022, pp. 579–596.
- [22] W. Xiao et al., "Gandiva: Introspective cluster scheduling for deep learning," in *Proc. OSDI*, 2018, pp. 595–610.
- [23] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 1–17.
- [24] K. Karanasos et al., "Mercury: Hybrid centralized and distributed scheduling in large shared clusters," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2015, pp. 485–497.
- [25] E. Boutin et al., "Apollo: Scalable and coordinated scheduling for {cloud-scale} computing," in *Proc. 11th USENIX Sym. Operating Syst. Des. Implementation (OSDI)*, 2014, pp. 285–300.
- [26] A. Van Den Oord et al., "WaveNet: A generative model for raw audio," *SSW*, vol. 125, pp. 1–12, 2016.
- [27] Slurm. "Slurm@ commercial support and development," Accessed: Aug. 23, 2023. [Online]. Available: <https://www.schedmd.com/>
- [28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
- [29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
- [30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 2818–2826.
- [31] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," 2016, *arXiv:1611.01603*.
- [32] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," 2016, *arXiv:1606.05250*.
- [33] "Wmt16 dataset." 2022. [Online]. Available: <http://www.statmt.org/wmt16/>
- [34] J. Yamagishi, "English multi-speaker corpus for CSTR voice cloning toolkit," 2012. Accessed: Aug. 23, 2023. [Online]. Available: <http://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html>
- [35] "Common voice dataset." 2022. [Online]. Available: <https://voice.mozilla.org/>
- [36] H. Zhao et al., "Coda: Improving resource utilization by slimming and co-locating DNN and CPU jobs," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*. Piscataway, NJ, USA: IEEE, 2020, pp. 853–863.
- [37] C. Olston et al., "Tensorflow-serving: Flexible, high-performance ML serving," 2017, *arXiv:1712.06139*.
- [38] W. Cui et al., "Enable simultaneous DNN services based on deterministic operator overlap and precise latency prediction," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2021, pp. 1–15.
- [39] AWS Lambda. "Lambda function scaling". Accessed: Aug. 23, 2023. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html>
- [40] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Serving heterogeneous machine learning models on {multi-GPU} servers with {spatio-temporal} sharing," in *Proc. USENIX ATC*, 2022, pp. 199–216.
- [41] M. Han, H. Zhang, R. Chen, and H. Chen, "Microsecond-scale preemption for concurrent {GPU-accelerated}-{DNN} inferences," in *Proc. OSDI*, 2022, pp. 539–558.
- [42] "Linux command tc." 2022. [Online]. Available: <https://man7.org/linux/man-pages/man8/tc.8.html>



**Han Zhao** received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in June 2022. He is an Assistant Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include high performance computing and resource management of accelerators in datacenters.





**Weihao Cui** received the B.Sc. degree from Shanghai Jiao Tong University, China. He is currently working toward the Ph.D. degree in the field of computer science under supervision of Dr. Quan Chen with the Department of Computer Engineering, Faculty of Shanghai Jiao Tong University, China. His research interests include high performance computing and resource management of accelerators in datacenters.



**Quan Chen** (Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in June 2014. He is a tenure-track Associate Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include high performance computing, task scheduling in various architectures, resource management in datacenter, runtime system, and operating system.



**Jingwen Leng** (Member, IEEE) received the Ph.D. degree from the University of Texas at Austin. He is a tenure-track Associate Professor with the Computer Science and Engineering Department and John Hopcroft Computer Science Center at Shanghai Jiao Tong University. He research interests include holistic approach in addressing the performance, efficiency, and reliability for heterogeneous computing systems.



**Deze Zeng** (Member, IEEE) received the Ph.D. and M.S. degrees in computer science from the University of Aizu, Aizu-Wakamatsu, Japan, in 2013 and 2009, respectively. He is currently an Associate Professor with the School of Computer Science, China University of Geosciences, Wuhan, China. His current research interests include network function virtualization, cloud computing, software-defined networking, data center networking, networking protocol design, and analysis.



**Minyi Guo** (Fellow, IEEE) received the Ph.D. degree in computer science from the University of Tsukuba, Japan. He is currently a Zhiyuan Chair Professor and the Head of the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include parallel/distributed computing, compiler optimizations, embedded systems, pervasive computing, big data, and cloud computing. He is now on the editorial board of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, and *Journal of Parallel and Distributed Computing*. He is a fellow of CCF.