CODA: Improving Resource Utilization by Slimming and Co-locating DNN and CPU Jobs

Han Zhao*, Weihao Cui*, Quan Chen*†, Jingwen Leng*, Kai Yu*, Deze Zeng‡, Chao Li*, Minyi Guo*†.
*_Department of Computer Science and Engineering, Shanghai Jiao Tong University_ Shanghai, China
†_Shanghai Institute for Advanced Communication and Data Science, Shanghai Jiao Tong University_ Shanghai, China
‡_China University of Geoscience_ Wuhan, China
{zhaohan_miven,weihao,chen-quan,leng-jw,kai.yu}@sjtu.edu.cn, deze@cug.edu.cn, {lichao,guo-my}@cs.sjtu.edu.cn

Abstract—**While deep neural network (DNN) models are often trained on GPUs, many companies and research institutes build GPU clusters that are shared by different groups. On such GPU cluster, DNN training jobs also require CPU cores to run pre-processing, gradient synchronization. Our investigation shows that the number of cores allocated to a training job significantly impact its performance. To this end, we characterize representative deep learning models on their requirement for CPU cores under different GPU resource configurations, and study the sensitivity of these models to other CPU-side shared resources. Based on the characterization, we propose CODA, a scheduling system that is comprised of an adaptive CPU allocator, a real-time contention eliminator, and a multi-array job scheduler. Experimental results show that CODA improves GPU utilization by 20.8% on average without increasing the queuing time of CPU jobs.**

I. Introduction

The training of a deep learning (DL) model is a notoriously difficult and time-consuming process. To address such a challenge, a large number of enterprises build their own private GPU clusters and share them between different groups to amortize the cost. This leads to the emergence of multi-tenant GPU cluster that runs both CPU Jobs and GPU jobs. For example, model training often requires enormous and expensive GPU resources, companies such as Facebook choose to run the model inference job on the CPU [1]. This multi-tenant GPU cluster represents a new paradigm of private Cloud and requires a dedicated study on its workload characteristics, resource utilization efficiency and optimization.

Training a DL model is a complex process that has frequent CPU-GPU interactions. The model training with multiple GPU nodes can be divided into four steps. 1) The GPUs use the data in their global memory for training, and the CPU-side worker simultaneously prepares the next batch of data. 2) Each GPU sends the calculated gradient to the parameter server (PS) on CPU and waits for the synchronization. 3) The PS gathers all the gradients, calculates the new gradients, and sends the updated DL models to all GPUs. 4) each GPU starts the next computation step with the new gradients. In this process, the data prefetch and the gradient update of the PS run on CPU, and can be affected by CPU-side jobs.

There are some prior works on characterizing or managing the GPU clusters for DNN training jobs, such as Microsoft's [2] and Google's [3]. Jeon et. al [2] pay more attention to task locality, error, and queuing time. The work directly splits all the CPUs and memory to all GPUs, and lead to underutilization of CPU resources. Kelp [3] shows that 16% of the nodes in Google's production TPU cluster experience peak bandwidth higher than 70% of the available bandwidth, and proposes to optimize the allocation of memory bandwidth. Although they consider bandwidth contention, they ignore the DL model's requirement on the CPU resources that significantly impact the performance of the training. This causes the low GPU utilization and low throughput of the cluster. Worse, since GPU could do more data transformation than TPU, Kelp's performance will be further diminished.

In this work, we perform a detailed analysis of the GPU job's demands and contention for CPU resources on a production-scale multi-tenant GPU cluster shared by several artificial intelligence startup companies and research institutions. The analysis reveals two key findings. As for the first finding, For more than 10,000 GPU jobs in the cluster, a GPU job often applies cores in a stable mode. A large number of GPU jobs apply for one core or two cores, and a considerable number of GPU jobs apply for over ten cores. Despite the stable application mode, our investigation shows that GPU jobs have different requirements for the cores to achieve the best performance. The CPU resource assigned to a DNN training job significantly impacts the job's performance (to be discussed in detail in Section IV). As for the second finding, adopting the widely-used FIFO or DRF job scheduling, the GPU cluster suffers from low GPU utilization, while both CPU and GPU jobs suffer from long queuing time on the contrary.

The two findings motivate us to propose **CODA**, a resource and job scheduling system that improves resource utilization while maximizing the performance of the DNN training jobs in multi-tenant GPU clusters. There are three challenges that have to be resolved in CODA. 1) The optimal number of CPU cores needed by DNN training jobs vary. CODA has to identify the optimal one at runtime. 2) the CPU-side work of DNN training jobs contend for the shared resources (e.g., last level cache and memory bandwidth), and the contention may result in severe performance degradation. CODA has to be able to monitor the contention on shared resources and schedule the DNN jobs accordingly to avoid serious shared resource contention on all the nodes. 3) GPU resource fragmentation

Quan Chen, Jingwen Leng, and Minyi Guo are the corresponding authors.

exists with FIFO and DRF scheduling. CODA has to be able to minimize the GPU resource fragmentation to improve system-wide performance.

To be more specific, CODA is comprised of an *adaptive CPU allocator*, a *real-time contention eliminator*, and a *multi-array job scheduler*. The adaptive CPU allocator finds the optimal CPU cores for a DNN training job. Based on the model type information, the allocator finds the optimal CPU number in less than four attempts. The contention eliminator monitors the memory bandwidth used by each CPU job in real-time. When the bandwidth of one CPU job exceeds the specified threshold, the eliminator throttles its bandwidth usage to avoid performance impact on GPU jobs. The multi-array scheduling module classifies CPU jobs and DNN training jobs, and schedule them differently. In the model, each array is pre-allocated a part of the CPU resources, jobs in different queues can initially only use resources in this own array. When a large number of CPU jobs are queued and the GPU task queue is idle, the CPU job array can preempt some CPU resources from GPU job array accordingly, and vice versa.

This paper makes the following main contributions:

- **Comprehensive analysis of CPU-side resource requirement of DNN training jobs.** The in-depth analysis enables the design of the CODA for maximizing the performance of DNN training jobs.
- **The design of a feedback-based adaptive CPU allocation algorithm.** It identifies the optimal number of CPU cores that should be allocated to a DNN training job.
- **The design of a multi-array job scheduling policy.** Based on the core requirements of DNN jobs and the shared resource contention, multi-array scheduling reduces the GPU fragmentation and improves system-wide performance.

Experimental results based on real-system job trace show that CODA improves the GPU utilization by 20.8% without degrading the queuing performance of low priority CPU jobs.

II. Related Work

In this section, we describe previous works on resource management in a datacenter. Previous work mainly focuses on three aspects, which is the design of the scheduling algorithm, the scheduling systems for specific scenarios, and the performance analysis of the cluster and jobs.

A. Scheduling algorithm

Previous researches have proposed different scheduling algorithms to solve the job scheduling problem on clusters. DRF [4], a generalization of max-min fairness to multiple resource types, addresses the problem of fair allocation of multiple types of resources to users with different demands. Mainstream job scheduling framework such as Yarn [5] and Mesos [6] all adopt it as one option of their scheduling choice. Choosy [7] is an extension to max-min fairness, which solves the problem of max-min fairness in the presence of constraints. Delay scheduling [8] is a simple strategy that when the job that should be scheduled cannot launch a local task, it waits for a small amount of time, letting other jobs launch tasks instead. This strategy is also used and adapted to many job scheduling frameworks for better throughput. These works are orthogonal to our work, because our work is mainly for the design of the scheduling system on the GPU cluster, rather than the optimization of the scheduling algorithm.

B. Scheduling system

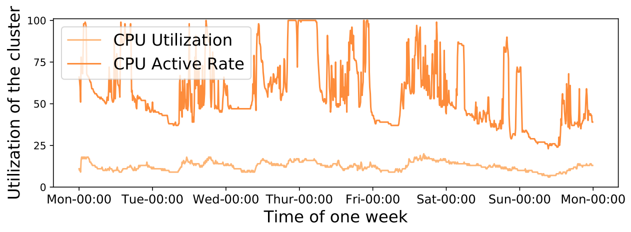
Many researches [9]–[15] aim to solve the scheduling problem in specific scenarios. Gandiva [11] analyzes the characteristics of the application by investigating the training characteristics of the application and finds the best hyper-parameter setting in the scenario of Auto-ML. And kelp [3] attempts to control the memory bandwidth allocation on the CPU side, thereby reducing the corresponding interference and improving the scheduling efficiency of the TPU cluster. Optimus [12] uses online performance models to find relative resource configuration to reduce training time. Baymax [13] and Prophet [14] adopt multitasking to improve the cluster utilization for the pure GPU workloads. Slaq [15] collects the quality improvement for resource usage and schedule concurrent machine learning jobs for average quality improvement. Since they are proposed for specific application scenarios, they do not work in the Multi-Tenant GPU cluster, which has also prompted us to conduct new research.

C. Analysis of cloud applications

One paper of Google [16] focuses on memory access analysis for specific applications of traditional CPU clusters while another Google paper [17] keeps an eye on distribution characteristics of jobs and the relationship between jobs and microstructures through more holistic cluster analysis. Recently, there have been some papers that focus on job analysis on GPU clusters. One research of MSRA [2] focuses on the topology of jobs and queuing time issues, and the reasons for the job error. Another research of Facebook [1] analyzes the distribution and characteristics of deep learning jobs on existing clusters and optimizes job management. These researches are instructive for the workload analysis under the GPU cluster. However, these papers are still complete enough, and they do not propose a system to solve the new problems in the GPU cluster.

III. Motivation for CPU Resource Management

In this section, we analyze the resource usage on a production-scale multi-tenant GPU cluster built for training DL models. The conventional wisdom is that GPU is the most critical resource for DL training. However, our analysis shows that CPU also plays a vital role in DL training. Naively allocating too few CPUs to the training job limits the training performance while allocating too many CPUs leads to significant queuing delay because CPU now becomes the scarce resource.



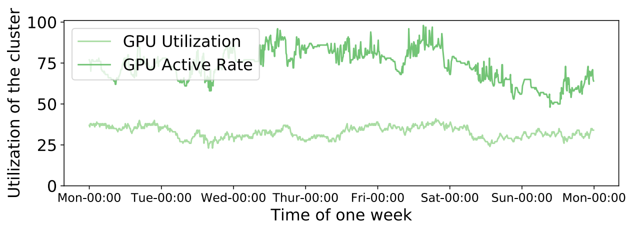


Fig. 1: The CPU and GPU utilization trend of the cluster through one week.

A. *Real-world GPU Cluster Investigation*

We conduct our investigation on a real-world multi-tenant GPU cluster[1]. The cluster comprises of about 80 PCIe-based multi-GPU (mostly GTX 1080Ti) servers. Each server has two sockets where the CPU is mostly Intel Xeon Gold 6132 with 14 cores. This cluster is shared by four AI startup companies and one AI research institution. Those companies work in the area of automatic speech recognition, natural language processing, and computer vision. The cluster has a centralized job management system SLURM [18] that uses FIFO to schedule jobs from different parties. Each job can request a certain number of CPU and GPU separately.

The cluster tenants include several artificial intelligence startup companies and research institutions, which frequently perform DL model training. We collect the cluster's CPU/GPU usage characteristics and job information through a week, which leads to a contradictory observation. **The training jobs do not fully utilize the cluster's GPU resources, but many jobs take long queueing time to get submitted.**

1) Resource Usage: We first analyze the resource (i.e., CPU and GPU) usage in the cluster, which we evaluate through the active rate and utilization rate. Equation (1) illustrates the active CPU rate, which is the ratio between actively used CPUs and the total number of CPUs. For an active CPU, the utilization rate indicates its time spent in the actual work and is collected via the operating system. We calculate the cluster's CPU utilization rate as the average across all active CPUs. The GPU related metrics are calculated similarly.

$$Active\ CPU\ Rate = Number\ of\ Active\ CPUs/Total\ CPUs \tag{1}$$

We collect the cluster's CPU and GPU usage through a week. As Fig. 1 shows, the GPU utilization is consistently higher than the CPU, which can be explained by the high computation requirement of training jobs. However, we also observe that the GPU exhibits a relatively stable active rate while the CPU exhibits the diurnal pattern. We dive into the job characteristics to search for its root cause.

2) Job Characteristics: We first categorize the jobs into GPU jobs and CPU-only jobs. Fig. 2a shows the job type breakdown. The AI research lab contributes to the most to the GPU jobs while the AI companies contribute most to the CPU jobs. The difference can be explained by the different parties' emphasis on the DL model training/inference. The research lab emphasizes the model training which heavily relies on the GPU while the AI companies emphasize the model inference, which typically uses the CPU [1]. Since the AI companies are user-facing and user requests are usually bursty, it also explains the diurnal pattern of CPU's active rate.

[1]A GPU cluster from AISpeech Co., Ltd.

TABLE I: Representative DNN models.

Neural model	Scenario	Type	Dataset
Alexnet [19]	CV	CNN	ImageNet [20]
VGG16 [21]	CV	CNN	ImageNet
InceptionV3 [22]	CV	CNN	ImageNet
Resnet-50 [23]	CV	CNN	ImageNet
Bi-att-Flow (BAT) [24]	NLP	RNN	SQUAD [25]
Transformer [26]	NLP	-	WMT16 [27]
Wavenet [28]	Speech	CNN	VCTK [29]
DeepSpeech [30]	Speech	RNN	Common Voice [31]

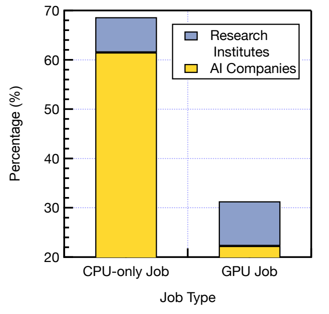
3) Queueing Delay: We now analyze the job queueing delay in Fig. 2c. We observe that the GPU jobs experience significantly more delay than the CPU jobs. About 48.1% GPU jobs waiting for at least 3 minutes and 41.3% GPU jobs waiting for more than 10 minutes. However, as Fig. 1 shows, the GPU's active rate is rarely more than 90%. In contrast, the CPU's active rate can reach 100% in the peak times.

We next study the requested CPU-GPU ratio for GPUs to understand the long queueing delay of GPU jobs. Fig. 2d shows that 15.3% of GPU jobs request more than 10 CPU cores. This leads to the insufficient CPU cores of these servers, and these servers cannot accommodate the incoming GPU jobs, which means the waste of CPU resources and GPU resources. Besides, 76.1% of jobs are requesting 1 CPU or 2 CPUs. Theoretically, the insufficient CPU will cause the slowdown of programs, which in turn lead to lower GPU utilization. As such, we can conclude that the CPU is the critical resource in the cluster as it is the root cause for the long queueing delay of GPU jobs.

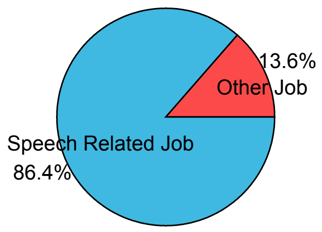
B. *DNN Training CPU Usage Analysis*

After recognizing the CPU as the important resource in the cluster, we now study *how the CPU core number affects the DL model training performance*. We conduct the study using a set of representative DL models from various domains. Our results show that the impact of CPU core number on the model training performance is highly model- and domain-specific. Allocating too few cores limit the single-model training performance while allocating too many cores limit the other jobs' performance. As such, in order to balance the single-model performance and the overall cluster utilization, we need an automatic and intelligent CPU core allocation strategy.

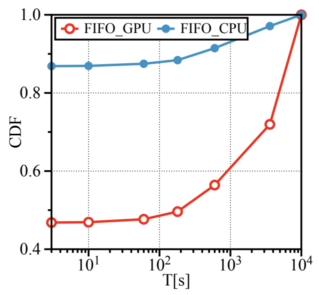
Tbl. I shows the set of representative DNN models. We select state-of-the-art DL models including DeepSpeech [30], Transformer [26], BAT [24], and Wavenet [28] that target



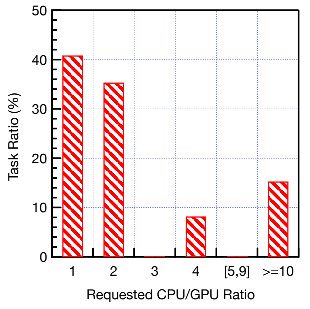
(a) Distribution of different job types.



(b) Percentage of speech-related jobs.



(c) CDF of job queuing time.



(d) The CPU/GPU ratio of GPU jobs.

Fig. 2: Information of CPU-only and GPU-based DNN training jobs.

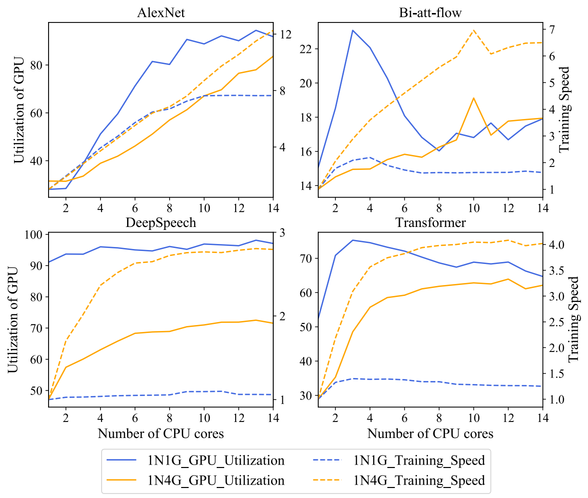


Fig. 3: The GPU utilization when the training job uses different numbers of CPU cores.

speech recognition, machine translation, question answering, and speech synthesis.

Fig. 3 shows the training speed and GPU utilization of various models with two training configurations (1N1G and 1N4G) when the core number increases. We have two observations from Fig. 3. The first is that the GPU also achieves the highest utilization at the maximal training speed. This observation matches our intuition as GPU computation is the major step in the training process. The second one is that most of the models do not gain the best performance with 2-CPU configuration except Transformer with 1N1G configuration. The performance gap is in the range of 10% to over 5X. The requirements of the benchmarks for CPU cores are different. The existing CPU allocation strategies result in the poor performance of the DNN training jobs.

IV. Analysis of CPU Requirement

In this section, we analyze the CPU-side resource requirements of mainstream models. We seek to answer three questions. (1) What is the best-fit CPU number for different deep learning GPU models? (2) What are the factors that determine the CPU demands of one model and how the factors determine its demand? (3) Is there any other CPU-side resource that the performance of GPU jobs is sensitive?

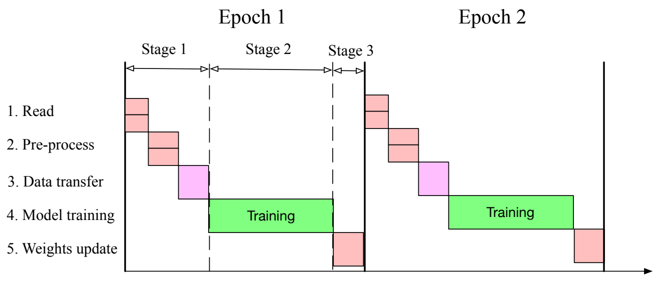


Fig. 4: The CPU-GPU collaborative process.

A. CPU-GPU Collaborative Process

Fig. 4 shows the collaborative process of CPU and GPU when training a DNN model using a single GPU. (1) Read data from disk into memory. (2) Pre-process raw data to proper format that can be used by 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 stage 4. Besides run the five steps in serial, programmers can parallelize stage 1, or pipeline stage 1 and stage 2 to further improve the performance.

The specific process steps of training models in different fields are slightly different. The second step of CV models converts the raw image into the pixel matrix used for training. The SPEECH models need to do interception and transformation of audio snippets at the same step, and it takes a longer time in transformation for a more complex operation. As for NLP models, since the mainstream datasets are relatively small, mainstream implementation of NLP models choose to read the entire dataset into memory which avoid the first step of the collaborative process. However, NLP jobs also need to do conversion work on the CPU side, such as converting one word to a one-hot vector.

B. CPU demands for GPU jobs

In this section, we use the notation of $aNbG$ to represent the training configuration of using a servers and b GPUs. Fig. 5

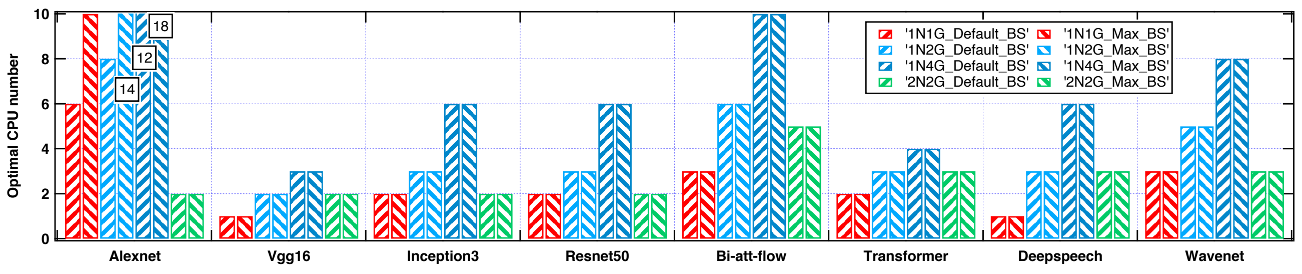


Fig. 5: The optimal CPU core number for different benchmarks with different batch size.

shows the optimal core number for the studied models with different configurations and batch size(BS).

1) Single-GPU CASE: Observed from Fig. 5, all models except Alexnet have the same CPU demands in the default BS configuration and the maximum BS configuration. This is because the increased BS increases both the data preparation time and the GPU computing time. When the GPU computing time is relatively long compared to the data preparation time, changes in the BS do not affect the computing requirements of the job for data processing.

For CV jobs, the simpler the network, the more CPUs are required. While the models with the same BS have the same data preparation time, more complex networks have longer GPU computing time. With pipeline optimization, more complex models require fewer CPUs to speed up data preparation time. For NLP jobs, although they do not need to read the data into memory, it needs to prepare the vectors required for training before each iteration. For example, BAT needs to prepare the context vector and query vector needed for the next iteration and Transformer needs data preprocessing, batching, and shuffling. For SPEECH jobs, the data preparation phase is similar to CV jobs, which require data reading and format conversion. Besides, Wavenet needs to re-cut the audio during the data preparation stage, and Deepspeech does not. So Wavenet needs more CPU cores than Deepspeech.

In single-GPU case, 1) CPU demands of most models are independent of BS. 2) The CPU demands of the CV models are mainly related to the model complexity. The higher the model complexity, the less CPU is required. 3) The CPU demands of the NLP models are mainly related to the data preprocessing between iterations. The more related vector calculations, the more CPU is required. 4) The CPU demands of the SPEECH models are mainly related to the design of the model. Whether additional audio processing is required determines the demand for the CPU.

2) Multi-GPU CASE: In the single-node multi-GPU configuration, the model's CPU demands have a linear relationship with the number of GPUs it requires. This is because its CPU-GPU interaction process is similar to the single-node single-GPU configuration. The impact of local communication on the overall process is small. The GPU number increase only affects the amount of computation in the data preparation stage, which results in a linear relationship between the model's CPU demands and the number of GPUs in the configuration. For

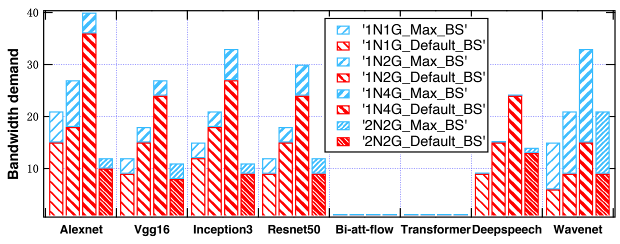


Fig. 6: The memory bandwidth demand for different benchmarks with optimal CPU number.

different models, due to the different computing requirements between iterations, the growth slope of the CPU demands of different models is different.

Under the multi-node multi-GPU configuration, the experimental configuration is a 10Gb/s Infiniband network interconnection. First, it can be seen that the CPU requirements of all models are no more than two cores. According to our observations, all models have 25% -30% performance degradation compared to 1N4G configuration. This is because the network communication in a multi-node configuration cannot be optimized. The decreased performance also leads to jobs consuming more time to prepare the data.

In multi-GPU case, 1) the model's CPU demand is still independent of the batch size. The reason is the same as the single-node single-GPU configuration. 2) if the GPUs are on the same node, the model's CPU demand is linearly related to the GPU number, and the relevant slope is determined by its data preprocessing requirement. 3) if the GPUs are on different nodes, the CPU demand of the model is reduced due to the impact of network communication. The CPU demand reduction is determined by the model weights and the network communication speed.

C. Analysis for CPU-side resource contention

The bandwidth usage here refers to the maximum memory bandwidth used during model training. The corresponding results are shown in the Fig. 6.

1) Memory bandwidth demand of GPU jobs: First, it is easy to find that memory bandwidth demand of CV models have an anti-correlation with its model complexity, which is consistent with the CV models' CPU demands. In general, when a model has lower complexity, it needs more CPU to complete the data preparation, which also means more need

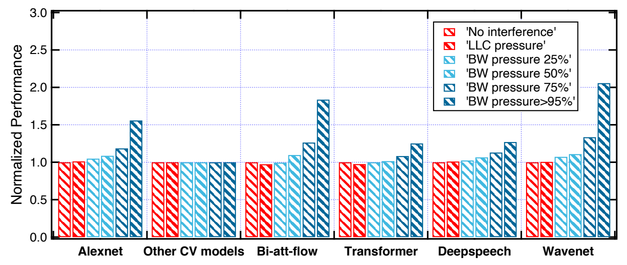


Fig. 7: The normalized performance of all the 1N1G models under contention.

for bandwidth. Note that, when the model adopts a larger BS, its bandwidth demand increases slightly. Second, For the NLP model, we can find that the bandwidth requirements of both NLP models are very small. There are two reasons for that. One is the small dataset. NLP models choose to read all the datasets into the memory, avoiding massive memory access operations between iterations. The other is the NLP models' input is pretty small such as the one-hot vector. Third, for the SPEECH models, we find that its bandwidth demand is also different due to its different data pre-process operations. With the BS increasing, the bandwidth requirement of Wavenet increase accordingly for its need for audio re-cut, while Deepspeech's demand does not change. Besides, when the configuration is increased, the bandwidth requirements of different models increase linearly, which is consistent with CV models. For the multi-GPU configuration, as the GPU number increases, the CV models and SPEECH models' memory bandwidth demand increases linearly.

2) Memory bandwidth contention of GPU jobs: To investigate the sensitivity of a DNN training job on the memory bandwidth contention, we have chosen a memory-intensive benchmark HEAT to inflict bandwidth pressure. By adjusting the thread number of the program, different levels of LLC or bandwidth pressure is applied to models on the same node. We only show the experimental results of the 1N1G models in Figure 7 due to space limitations.

Observed from Fig. 7, all the models are not sensitive to the LLC contention. Meanwhile, for CV models, only Alexnet is affected by the memory bandwidth contention, while Vgg16, Inception3, and Resnet50 are insensitive to the bandwidth contention. Since these models require less memory bandwidth and they have longer running time for every iteration, they are insensitive to the memory bandwidth contention. NLP models are more sensitive to the memory bandwidth contention. There have been at least a 50% performance drop, which is consistent with the analysis of previous sections. While NLP models all need complex data preprocessing between iterations, they are sensitive to the resources contention, including memory bandwidth contention and other contention like bus introduced by bandwidth contention. For SPEECH models, Deepspeech is more sensitive than Wavenet, for it needs fewer data preprocessing between iterations.

In summary, DNN models have different sensitivity to memory bandwidth contention.

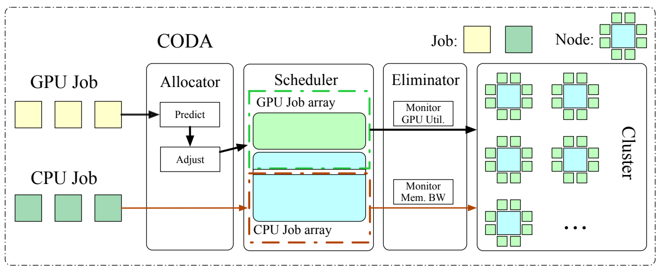


Fig. 8: Design overview of CODA.

3) PCIe bandwidth contention: Our experiment shows that all models do not consume more than half of the bandwidth of PCIe 3.0, which is 16GB/s (detailed data is omitted due to the limited space). In other words, the co-running of two models do not exceed the total available PCIe bandwidth. As such, it is safe to co-locate two $1N1G$ training jobs in a single node as it does not cause performance degradation for training. As for the $1N2G$ case, the noticeable performance drop exists only when one of the co-located training jobs is Alexnet or Resnet50. Their maximum PCIe bandwidth consumption is 12 GB/s, with the averaged value of 8 GB/s. In contrast, NLP and speech models only consume less than 1 GB/s of the PCIe bandwidth. The difference in the PCIe bandwidth consumption explains why the CV models can cause large performance drops when co-located with other models.

Therefore, the selected models in this paper require little PCIe bandwidth. Unless the complexity of the model is small, it will demand a considerable amount of PCIe bandwidth. Based on the contention experimental results, it can be seen that the co-running of two low demand models does not affect each other. When co-running with a higher PCIe demand model, there will be 5%-10% performance degradation.

V. Methodology of CODA

In this section, we present CODA, a scheduling system that improves resource utilization of multi-tenant GPU clusters by slimming and co-locating DNN training jobs and CPU jobs.

A. Overview

Fig. 8 shows the design overview of CODA that schedules both CPU jobs and DNN training jobs on a GPU cluster. As shown in the figure, CODA is comprised of an *adaptive CPU allocator*, a *real-time contention eliminator*, and a *multi-array job scheduler*. The CPU allocator identifies the optimal CPU number for a DNN training job at runtime. The contention eliminator watches the shared resource contention on each node. It ensures that the performance of a DNN training job would not be severely degraded by the contention on CPU-side shared resources. Based on the data provided by the CPU allocator, the job scheduler allocates CPU and GPU resources aiming for high performance. Besides the above three parts, CODA periodically updates the job information from all users and array-level job information in the backend. The information is useful for the CPU allocator and job scheduler to act efficiently.

Specifically, CODA 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 based on its category (Speech, CV, or NLP) and the owner's historical job information recorded in the log. 2) Job J is pushed into either the CPU job array or GPU job array based on its resource requirement. 3) The job scheduler assigns J the required resources based on the status of all the nodes in the cluster. 4) The contention eliminator on each node keeps monitor the memory bandwidth usage of each job and throttles the memory bandwidth of a CPU job if it consumes high memory bandwidth. 5) When J completes, its resource usage, scheduling information, and owner information are recorded in a log for future use.

Note that, in our current design, the contention eliminator of CODA only throttles the memory bandwidth usage of CPU jobs. The main reason for this design is that DNN training jobs have higher priority than all CPU jobs on GPU clusters except the user-facing inference jobs. Besides, the DNN training jobs themselves on the same node would not contend for memory bandwidth severely, as we showed in Section IV-C.

B. Adaptive CPU allocation

For the DNN training job J, the adaptive CPU allocator identifies its optimal core number based on two findings in Sec. III. First, a DNN training job's GPU utilization rate and running speed change in a similar trend, and they reach the optimal value at the same CPU number. Second, there is a linear relationship between the GPU utilization and the CPU number allocated to the job.

As shown in Fig. 3, for job J, if the number of its cores exceeds the optimal number, the corresponding GPU utilization of J drops slightly. To identify the optimal core number for J, the CPU allocator first finds a reasonable core number as the start point N_{start} to perform the search. The start point is determined based on the core number information of the owner's historical jobs and the category of J (Speech, CV, or NLP). The allocator then increases or decreases the number of cores allocated to J to check whether more or fewer cores would result in a higher performance of J.

1) Determining N_{start}: The CPU allocator determines the start point to perform the search according to the CPU-GPU interaction principle. According to the discussion in Sec. IV-A, the models of the same category have similar CPU-GPU processing modes. We assume that the tenants provided at least the categories of their models, and may provide the following three types of information: the number of model weights, 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 historical 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 empirically based on our investigation in Sec. IV-B. In the worst case that the owner of J does not

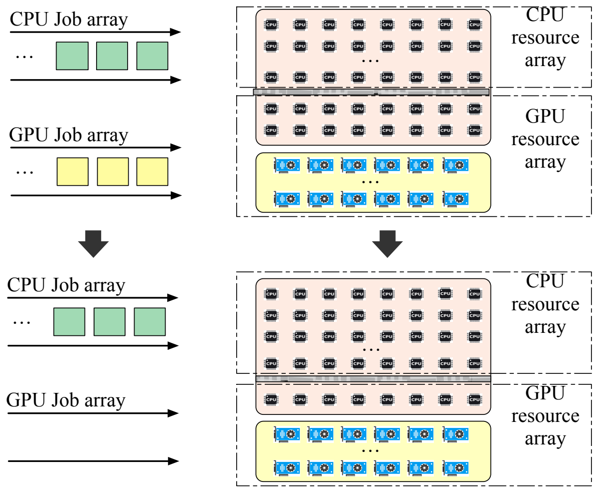


Fig. 9: The design of the multi-array job scheduling.

even provide the category of J, it is also sufficient to find a reasonable N_{start} based only on the owner's historical job execution information.

If the owner of J provides extra information, the N_{start} can be further optimized based on the investigation in Sec. IV-B. To be more specific, if J is implemented using pipeline optimization, the corresponding 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.

2) Tuning the core number: Starting from N_{start}, CODA tries both larger and smaller core number allocated to J (denoted by N_{opt}), and checks whether they speedup J. The CPU allocator first evaluates the smaller core number for J. Three cases may happen here. 1) If the GPU utilization improves, the allocator reduces N_{opt} until the GPU utilization does not improve. 2) Otherwise, if the smaller core number does not improve the GPU utilization, the allocator increases N_{opt} in the opposite direction. If larger N_{opt} improves the GPU utilization, the allocator keeps increases N_{opt} until the GPU utilization does not improve. 3) If the GPU utilization is not improved with both fewer or more cores, the most suitable number of cores for J is found. We analyze the effectiveness and related overhead of the tuning in Sec. VI-F.

C. Multi-array job scheduling

FIFO that gives the highest system-wide throughput and DRF that enforces fairness between the tenants are the two most widely-used scheduling algorithms in multi-tenant clusters. However, they result in three critical problems on a GPU cluster. First, FIFO and DRF may leave GPU jobs pending when a large number of CPU jobs are in the array. Second, a tenant that uses GPUs reaches a large weight with DRF, because GPU in the cluster is more scarce than CPU. The large weight makes it impossible for the tenant to submit CPU jobs in a fair time. Third, GPU jobs cannot be submitted if there are many GPU jobs that apply for one or two GPUs in the job array. In other words, the cluster has encountered GPU fragmentation caused by CPU jobs and GPU jobs, and there are corresponding fairness issues at the same time.

As shown in Figure 9, the multi-array job scheduler resolves the three problems by dividing all resources into CPU resource array and GPU resource array. The GPU resource array reserves some CPU resources for GPU jobs in this array. This part of the computing resources is derived from historical statistical information. For CPU job array, DRF scheduling is used to schedule the CPU jobs based on the usage of CPU. For GPU jobs in the GPU job arrays, DRF scheduling is used to schedule them according to the usage of GPU, and their CPU numbers are designated by the adaptive CPU allocator.

If CPU jobs burst and the GPU resource array is relatively idle, the multi-array scheduler allows CPU jobs to preempt the reserved cores in the GPU resource array (Figure 9). When a GPU job arrives and needs the preempted CPU cores, CODA aborts the running CPU job and releases the preempted CPU cores. The suspended CPU job re-enters the array head, waiting to be rescheduled again. Benefit from containerization and virtualization, similar to Container and Kata, job migration is easy to achieve here.

The GPU job array is further divided into a 4-GPU sub-array and a 1-GPU sub-array. The 4-GPU sub-array is for jobs that apply for 4 GPUs or more, while the 1-GPU array serves jobs that demand less than 4 GPUs. The division of the corresponding array is also determined by the statistical information of the historical jobs. The maximum GPU number required by 4-GPU jobs in the historical statistics is designated as the corresponding initial resource division. When the resources in the 4-GPU array are all used, the GPU job in the array will be allocated to the nodes that meet the requirements in the 1-GPU array. If no suitable node is found, the job queues for scheduling later. Similarly, if all the resources in the 1-GPU array are used, the job tries to preempt resources from the 4-GPU job array. When 4-GPU jobs need to use corresponding resources again, job migration is performed.

The multi-array design eliminates the problems of GPU fragmentation and the unfair resource usage of the users.

D. Real-time contention elimination

Based on the analysis in Sec. IV-C, DNN training jobs are sensitive to the contention on memory bandwidth.

In order to guarantee the performance of the high priority DNN training jobs, CODA 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) and the GPU utilization of the DNN training jobs on the node drops, the performance of the DNN training jobs is degraded due to the memory bandwidth contention.

In this scenario, the contention eliminator uses the Memory Bandwidth Allocation (MBA) technique to throttle the memory bandwidth of CPU jobs. 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, CODA tries to schedule new CPU jobs and uses the same method to ensure that the utilization of GPU is not affected.

VI. Evaluation of CODA

In this section, we evaluate the performance of CODA in improving the resource utilization of GPU clusters.

A. Reducing queuing time

We collected the task trajectory information for one month from the real reproduction cluster. There are 100, 000 jobs are submitted during the one month, in which 75,000 jobs are CPU jobs, and 25,000 jobs are DNN training jobs. Most of the GPU jobs are training NLP and SPEECH models, and there are also a large number of CPU jobs to complete some auxiliary tasks and other tasks. The specific experimental configuration has been explained in the Sec. III, which has 80 nodes and 400 NVIDIA 1080Ti GPUs.

Based on the above real-system traces, we compare CODA with two widely-used job scheduling algorithms for large-scale clusters: FIFO (First In First Out) scheduling policy, and DRF (Dominant Resource Fairness) scheduling policy. With FIFO, the CPU and GPU jobs are scheduled in the FIFO manner. With DRF, we consider GPU as the dominant resource and enforce that the tenants fairly share the dominant resource.

B. Improving resource utilization

Fig. 10 shows the GPU active rate and the GPU utilization with FIFO, DRF, and CODA. GPU active rate is the percentage of the overall GPUs that are used, and GPU utilization is the average resource utilization of each GPU. Higher GPU active rate means that more GPUs are utilized (shorter queuing time and lighter fragmentation). Higher GPU utilization means that each GPU is better utilized in training (higher training performance).

Observed from Fig. 10, CODA significantly improves the GPU utilization compared with FIFO and DRF. The GPU utilization of the cluster with FIFO, DRF, and CODA are 45.4%, 44.7%, and 62.1%, respectively. CODA improves the GPU utilization of the GPU cluster by 62.1%-45.4%=16.7%. At the same time, when the jobs queue up for the resource allocation, the GPU active rates of the cluster with FIFO, DRF, and CODA are 83.5%, 83.3%, and 91.2%, respectively. This shows that CODA improves overall GPU utilization.

The improved GPU utilization originates from the adaptive CPU allocation and real-time contention elimination in CODA. The adaptive CPU allocator in CODA 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.

By comparing the GPU active rate curve of CODA with the GPU active rate curve of FIFO in Fig. 10, we can find

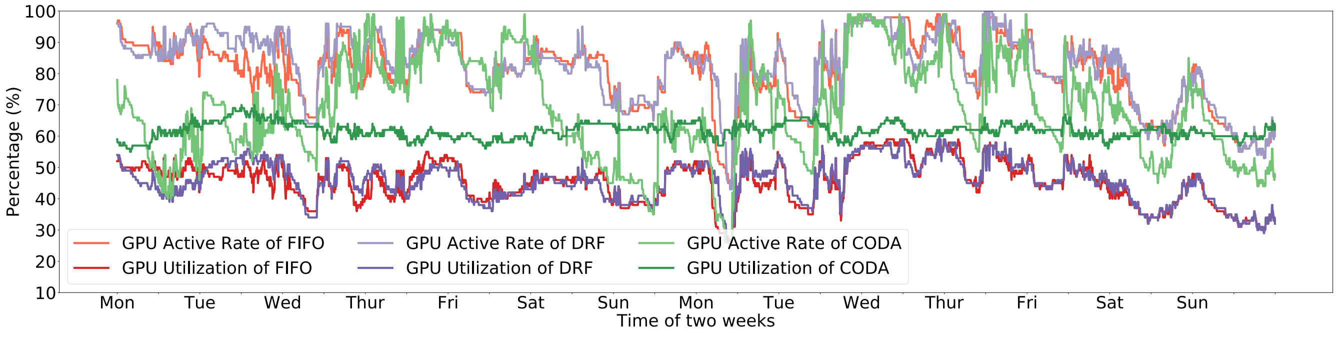


Fig. 10: The GPU active rate and GPU utilization of the multi-tenant GPU cluster with CODA, FIFO, and RDF.

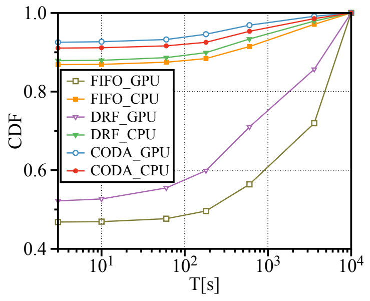


Fig. 11: The CDF of the job queuing time with FIFO, DRF, CODA.

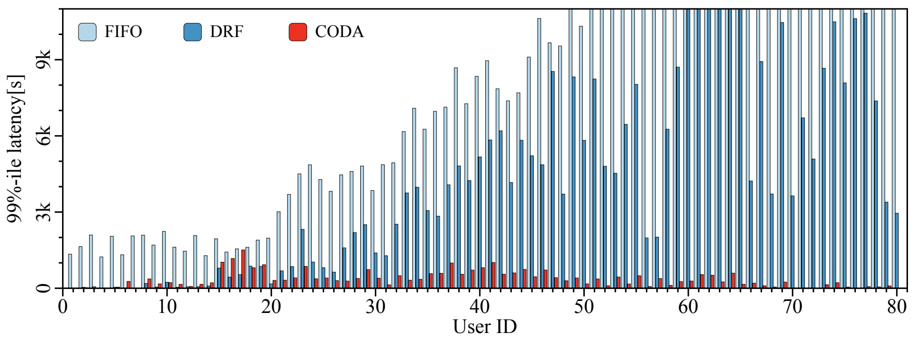


Fig. 12: The 99%-ile queuing time of each user with FIFO, DRF, and CODA.

that CODA reacts to the job change in advance. CODA first reduces GPU fragmentation to ensure GPU active rate and reduce task queuing time. Secondly, CODA further reduces the task queuing time by optimizing the task's running time, which also improves the throughput of the cluster.

C. *Eliminating GPU fragmentation*

GPU fragmentation happens in two cases. In the first case, GPUs are not used because there is not sufficient CPU for the DNN training job on the node. In the second case, a node is not able to host GPU jobs that require four GPUs or more if other GPUs on the node have already been allocated. We mainly discuss GPU fragmentation due to insufficient CPU here, while the second case is solved by the multi-array job scheduler in CODA.

According to our experimental results, the average fragmentation rate of CODA is less than 1%, while the fragmentation rate of FIFO and DRF is 14.3% and 14.6%. For FIFO and DRF, a large number of GPU jobs queue up for scheduling, while more than 14% of the GPUs are not used.

The advantages of CODA come from two aspects, On the one hand, it solves the problem of queuing and GPU fragmentation caused by tasks that apply for excessive CPU. On the other hand, the multi-array scheduling module ensures that GPU tasks are not affected by CPU tasks. Based on these two advantages, CODA significantly reduces GPU fragmentation on existing clusters.

Fig. 11 and Fig. 12 show the CDF of the job queuing time of all the jobs, and the 99%-ile job queuing time of each user with FIFO, DRF, and CODA. With FIFO and DRF, 43.1% and 28.9%of GPU jobs suffer from queuing time more than ten minutes, 27.8% and 14.3% of GPU jobs queue up for more than an hour. Besides, 87.4% and 87.8% of the CPU jobs can get resource allocation within 10 seconds. With CODA, 92.1% of GPU jobs can get resource allocation without queuing, and 94.5% of CPU jobs can be scheduled to the cluster within 3 minutes.

The short queuing time of CODA comes from its CPU allocation optimization and multi-array scheduling. First, The CPU allocation optimization solves the problem of excessive CPU application from GPU jobs, thereby reducing GPU fragmentation. Second, through the multi-array design, CODA minimizes the impact of bursty CPU jobs.

Observed from Fig. 12, the queuing time with FIFO for most users is longer than that with DRF. The root cause is that DRF aims for fairness, and FIFO aims for higher throughput. When a small number of users submit a large number of jobs, FIFO does not take into account the fairness of the users. The decision causes a large number of users to queue up and have a longer queuing time. DRF provides better fairness, users who submit a large number of jobs have longer queuing time, waiting for tasks of users applied for fewer resources. For CODA, the queuing time for all users is much shorter than FIFO and DRF. As described earlier, CODA reduces GPU fragmentation and improves task performance, which improves system throughput and optimizes task queuing time. Besides, CODA also guarantees fairness among users since the DRF algorithm is used for scheduling inside each array. As can be seen from the figure, users that submit more tasks have a longer queuing time.

Note that, when the users that only submit CPU tasks (id: 15-20) try to submit a large number of CPU tasks, the

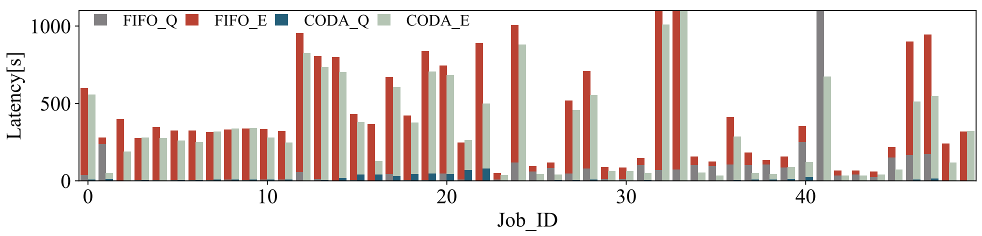


Fig. 13: The end-to-end latencies of representative GPU jobs with FIFO and CODA.

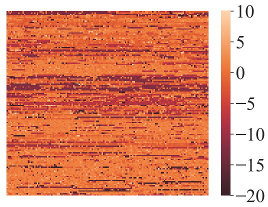


Fig. 14: Tuning the number of cores allocated to GPU jobs.

corresponding queuing time increase. This is because part of the CPU resources is reserved for the GPU task array, which results in a slightly longer queue time for these users. However, the corresponding queuing time is still not much different from the DRF, which means it is tolerable.

D. Effectiveness of tuning CPU allocation

Fig. 14 presents the tuning of the core number allocated to each DNN training job with CODA. 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 Sec. 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. 13 shows the end-to-end latency of representative GPU jobs (including queuing time and processing time). For each job, the left bar and right bar represent its queuing time and processing time with FIFO and CODA, respectively. Observed from the figure, CODA reduces the queuing time and processing time of most jobs simultaneously. The queuing time reduction originates from the improvement of the overall cluster throughput, and the processing time reduction originates from the CPU adjustment of the adaptive CPU allocator. CODA does not reduce or even slightly increase the processing time of a small number of GPU jobs. This is because when the processing time is very short, the corresponding benefits that the adaptive CPU allocation brings is not enough to cover up the overhead incurred by the module. Since CODA reduces the corresponding queuing time, the overall end-to-end latency of the GPU jobs still reduces.

In summary, the adaptive CPU allocation in CODA successfully tunes the core number for most DNN training jobs.

E. Effectiveness of eliminating CPU-side contention

The contention eliminator monitors the memory bandwidth usage of each node in real-time. Each bandwidth-intensive program was detected, and its bandwidth usage was restricted accordingly. In order to evaluate the effectiveness of the contention eliminator in CODA, we disable the eliminator in CODA and test the performance again. Our experimental results show that the average GPU utilization decreased by 2.3% when tasks are queuing, if the contention eliminator is disabled. Meanwhile, the overall number of queueing tasks doubles.

TABLE II: Overhead of identifying the optimal core number.

Neural model	Profiling steps	Training iterations
Alexnet [19]	4	about 260
VGG16 [21]	4	about 70
InceptionV3 [22]	3	about 180
Resnet-50 [23]	3	about 150
Bi-att-Flow [24]	4	about 35
Transformer [26]	3	about 260
Wavenet [28]	3	about 28
DeepSpeech [30]	3	about 45

The above performance data is reported in the scenario that only 0.5% of CPU tasks have high memory bandwidth requirements. If more CPU jobs on the cluster have higher memory bandwidth requirements, the performance is worse without the contention eliminator.

Memory bandwidth-intensive CPU jobs degrade the performance of DNN training jobs, without the contention eliminator. First, it affects the GPU utilization on the current node and the performance of the tasks running on the node. With the corresponding tasks' performance degrades, the number of queueing tasks in the cluster increases. Besides, because the memory bandwidth contention only affects the performance of GPU jobs on the current node, It does not affect the overall scheduling decision, so the corresponding cluster fragmentation rate does not change.

F. Overhead of identifying the appropriate core number

When identifying the appropriate core number for a DNN training job, CODA profiles the performance of the job with different core numbers. During the profiling, we sample the GPU utilization for each profiling step that lasts 90 seconds. As shown in Tbl. II, CODA identifies the optimal core number for all the DNN training jobs in 4 profiling steps, that last for 6 minutes. The table also lists the number of iterations that each model has been trained during the profiling step. Each model is trained for 28-260 iterations, which means that it is enough to discover the CPU requirements of jobs. Besides, we analyze the runtime distribution of GPU tasks in a week, and find that 39.6% of the DNN training jobs run for more than 2 hours and 68.5% of the training jobs run for more than an hour. It is worthwhile to spend about six minutes of exploration time to find the optimal core number.

G. Generailty

Some larger private clusters maybe composed of both GPU nodes and CPU nodes. For these more complex clusters, FIFO scheduling still results in low GPU utilization and GPU fragmentation. In addition to these problems, DRF faces new problems. When GPU resources are more scarce compared to CPU resources, a tenant that submits both CPU jobs and GPU jobs can easily reach a large weight. Then its CPU jobs would no longer be scheduled. This situation conveys unfairness among users. When GPU resources are sufficient relative to CPU resources, the weight of GPU becomes irrelevant. However, the multi-array scheduling in CODA ensures that the scheduling of GPU and CPU jobs is not affected by each other.

As for the clusters that have different workload characteristics, CODA also has excellent performance. The current mainstream private cloud is often adopted to do model training, which takes more than one hour or two hours. Although this paper does not cover all the models, GPU utilization represents the running speed of jobs theoretically. Therefore, CODA could find the best-fit CPU number of GPU jobs and ensure that GPU jobs all have the best performance. Further, CODA improves the throughput and utilization of the cluster, which means that CODA has good generality.

VII. Conclusion and Future Work

A multi-tenant GPU cluster hosts both DNN training jobs and traditional CPU jobs. We first characterize the CPU-side resource demand and contention of training DNN models in Speech, CV, and NLP field. Based on the analysis, we propose CODA, a scheduling system that improves the resource utilization of GPU clusters. CODA adopts a feedback-based method to find the just-enough core number for a DNN training job. Based on the analysis on the impact of contention on CPU-side resource on the performance of DNN training jobs, CODA eliminates the interference which ensures the performance gains of jobs. A multi-array job scheduler in CODA eliminates the GPU fragmentation. Experimental results show that CODA improves the GPU utilization by more than 20.8% without degrading the performance of GPU jobs.

Acknowledgment

This work was supported by the National Key R&D Program of China under Grant 2019YFF0302600, the National Natural Science Foundation of China (NSFC) (61632017, 61872240, 61832006, 61702328)

References

[1] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro *et al.*, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *HPCA*. IEEE, 2018, pp. 620–629.

[2] 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," *arXiv preprint arXiv:1901.05758*, 2019.

[3] H. Zhu, D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and M. Erez, "Kelp: Qos for accelerated machine learning systems," in *HPCA*. IEEE, 2019, pp. 172–184.

[4] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *NSDI*, vol. 11, no. 2011, 2011, pp. 24–24.

[5] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *SOCC*. ACM, 2013, p. 5.

[6] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: Flexible resource sharing for the cloud," *USENIX Magazine*, 2011.

[7] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *Eurosys*. ACM, 2013, pp. 365–378.

[8] 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 *Eurosys*. ACM, 2010, pp. 265–278.

[9] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "{HUG}: Multi-resource fairness for correlated and elastic demands," in *NSDI*, 2016, pp. 407–424.

[10] D. Crankshaw, G.-E. Sela, C. Zumar, X. Mo, J. E. Gonzalez, I. Stoica, and A. Tumanov, "Inferline: Ml inference pipeline composition framework," *arXiv preprint arXiv:1812.01776*, 2018.

[11] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *OSDI*, 2018, pp. 595–610.

[12] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *EuroSys*. ACM, 2018, p. 3.

[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] 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 Systems Review*, vol. 51, no. 2, pp. 17–32, 2017.

[15] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "Slaq: quality-driven scheduling for distributed machine learning," in *SOCC*. ACM, 2017, pp. 390–404.

[16] G. Ayers, J. H. Ahn, C. Kozyrakis, and P. Ranganathan, "Memory hierarchy for web search," in *HPCA*. IEEE, 2018, pp. 643–656.

[17] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 158–169, 2016.

[18] Slurm. [Online]. Available: https://www.schedmd.com/

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*. Ieee, 2009, pp. 248–255.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016, pp. 2818–2826.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[24] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," *arXiv preprint arXiv:1611.01603*, 2016.

[25] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[27] Wmt16 dataset. [Online]. Available: http://www.statmt.org/wmt16/

[28] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio." *SSW*, vol. 125, 2016.

[29] J. Yamagishi, "English multi-speaker corpus for cstr voice cloning toolkit," *URL http://homepages. inf. ed. ac. uk/jyamagis/page3/page58/page58. html*, 2012.

[30] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[31] Common voice dataset. [Online]. Available: https://voice.mozilla.org/