This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

Deep Learning Research and Development Platform: Characterizing and Scheduling with QoS Guarantees on GPU Clusters

Zhaoyun Chen, Wei Quan, Mei Wen, Jianbin Fang, Jie Yu, Chunyuan Zhang and Lei Luo

Abstract—Deep learning (DL) has been widely adopted in various domains of artificial intelligence (AI), achieving dramatic developments in industry and academia. Besides giant AI companies, numerous small-and-medium-sized enterprises, institutes, and universities (EIUs) have focused on the research and development (R&D) of DL. Considering the high cost of datacenters and high performance computing (HPC) systems, EIUs prefer adopting off-the-shelf GPU clusters as a DL R&D platform for multiple users and developers to process diverse DL workloads. In such scenarios, the scheduling of multiple DL tasks on a shared GPU cluster is both significant and challenging in terms of efficiently utilizing limited resources. Existing schedulers cannot predict the resource requirements of diverse DL workloads, leading to the under-utilization of computing resources and a decline in user satisfaction. This paper proposes GENIE, a QoS-aware dynamic scheduling framework for a shared GPU cluster, which achieves users' QoS guarantee and high system utilization. In accordance with an exhaustive characterization, GENIE analyzes the key factors that affect the performance of DL tasks and proposes a prediction model derived from lightweight profiling to estimate the processing rate and response latency for diverse DL workloads. Based on the prediction models, we propose a QoS-aware scheduling algorithm to identify the best placements for DL tasks and schedule them on the shared cluster. Experiments on a GPU cluster and large-scale simulations demonstrate that GENIE achieves a QoS-guarantee percentage improvement of up to 67.4% and a makespan reduction of up to 28.2%, compared to other baseline schedulers.

Index Terms—DL Research and Development Platform, Characterizing, Scheduling, QoS-aware, GPU clusters.

1 Introduction

Over the past few years, deep learning (DL) has been proven as a general and effective tool for diverse artificial intelligence (AI) fields [1, 2]. Emerging DL cloud services [3–5], provided by giant AI companies, further expand the adoption of DL in various business-critical processes. Besides, more and more small-and-medium-sized enterprises, institutes and universities (EIUs) are likewise focusing on the research and development (R&D) of DL, such as innovations in network structure, new application development and prediction accuracy improvement [6, 7], as shown in Fig. 1(a).

Different from DL cloud services provided by giant AI companies, DL R&D platforms among EIUs have their own specific features: 1) DL platform of giant companies always provided custom-designed servers, storage, and networking support for the resource requirement of each major workload [8–10]. Considering the high cost of customization, EIUs prefer adopting cost-effective commodity GPUs to assemble and build a limited-scale cluster to process diverse DL workloads. Therefore, a high-efficient scheduling for EIUs' DL platforms is extremely significant. 2) Most giant companies propose their own complete automation DL platforms which consist of a mix of technique optimizations and supports, such as HDFS, Spark, MLLib and TensorFlow [11, 12]. Due to the high cost of the development and maintenance of an automation DL platform, a decoupling scheduling framework, which is more suitable for EIUs, can minimize the changes of existing system and reduce the dependence on other techniques. 3) A highly specialized platform of giant companies unifies users' interfaces to support diverse workloads. All users should refer to the interfaces to customize own tasks [12, 13]. However, EIUs DL R&D platform only process diverse DL tasks (i.e., training and inference), providing support for a broad range of DL applications. Meanwhile, the scheduling frameworks for EIUs' scenarios try not to change the way of users' programming and task submitting, achieving high-efficient deployment. These specific EIUs' scenarios need customized scheduling design to improve processing efficiency.

However, most of the studies on this topic have so far focused on the requirements of giant companies concerning datacenters and high performance computing (HPC) systems, such as Tianhe supercomputer [14]. There is a small amount of research that has considered DL R&D platforms for EIUs. Prior work [15, 16] has studied the allocation of resources based on historical information and heuristics, where DL applications are scheduled simply as another big-data job, ignoring domain-specific knowledge. Taking **Graphite Cluster** of Cornell University [17] as an example, the cluster adopts the traditional SLURM for resource management, which cannot make full use of heterogeneous resources for diverse DL tasks. Load imbalance and resource under-utilization also appear in other existing cluster systems of numerous universities and laboratories [18–20], which leads to an enormous waste of resources.

• *Z.Chen, W.Quan, M.Wen, J.Fang, J.Yu, C.Zhang and L.Luo are with the Department of Computer, National University of Defense Technology, Changsha, China*
All correspondence should be addressed to L. Luo.
E-mail: l.luo@nudt.edu.cn

Manuscript received XXXX xx, 20xx; revised XXXX xx, 20xx.

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

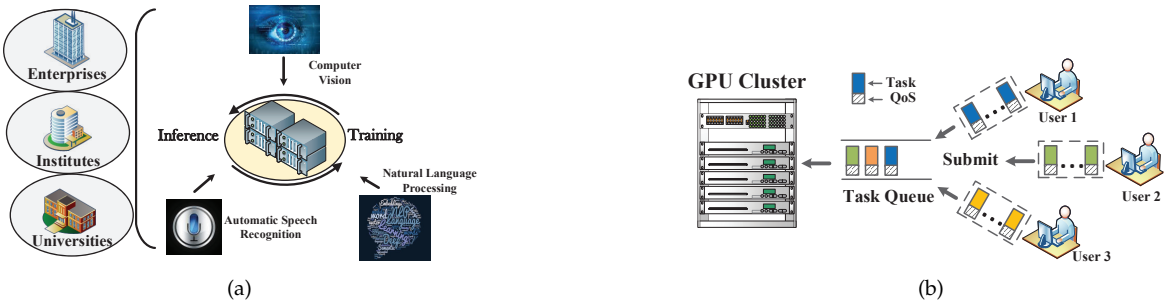


Fig. 1. (a) Small-and-medium enterprises, institutes and universities (EIUs) focus on the DL research and development (R&D) platform to process diverse DL tasks, supporting a wide range of AI applications. (b) A GPU cluster is widely adopted as the DL R&D platform. Multiple users and developers submit their DL tasks with various QoS requirements to the shared GPU cluster.

Recently, there are some studies that have focused on DL task scheduling for the purpose of exploiting cluster capability. Researchers [21, 22] have focused on scheduling design only for batch inference, which is more suitable for online DL services than DL R&D platforms. Researchers [23, 24] optimized and accelerated only a single DL training task on a distributed cluster, achieving the fastest training speed. Meanwhile, researchers [6, 7, 25, 26] have also attempted to explore and optimize multiple training tasks co-located on a server or cluster, but without QoS consideration. As shown in Fig.1(b), different users and developers submit their own DL tasks with various configurations and QoS requirements to a limited scale DL R&D platform. In these real DL R&D scenarios, it is critical to fulfill **users' QoS guarantees** and **high system utilization** at the same time, which is meaningful and challenging.

To maximize the system utilization and guarantee users' QoS requirements, a scheduler should be able to utilize the domain-specific knowledge of DL workloads to predict the relationship between resource provision and its impacts on task performance. It should thereby be able to make decisions concerning resource allocation and task placement according to the current conditions of system workloads. Thus, in this paper, we focus on DL R&D scenarios and attempt to propose a novel scheduling solution to fulfill the targets mentioned above. Our work aims to answer the following four research questions:

- Which factors among diverse application configurations and task placements have the largest impact on processing performance for DL tasks?
- How should the proper models for off-the-shelf GPU clusters and diverse DL tasks be constructed?
- In DL R&D scenarios, how should users' QoS and system utilization be quantified? How should the scheduling objectives for these scenarios be designed?
- How might resources be allocated dynamically for multiple DL tasks to effectively exploit the GPUs, achieving the above scheduling objectives?

To answer these questions, we propose GENIE, a QoS-aware dynamic scheduling framework to process multiple DL tasks on a GPU cluster. In more detail, the major contributions of this paper to the field are as follows:

- Focusing on GPU clusters adopted as DL R&D platforms in most EIUs, which are not considered much in the most current research in this area, we present a comprehensive characterization for DL tasks and show the performance impact of various application configurations and task placement policies.
- Based on observations from the characterization, we provide an analytical model to quantify the relationship of resource provision and processing rate through a lightweight profiler.
- According to the analytical model, we design a QoS-aware dynamic scheduling framework GENIE to identify effective placements for DL tasks on GPU clusters.
- An evaluation on a real GPU cluster is presented, demonstrating that GENIE outperforms other baselines in terms of QoS-guarantee percentage and makespan. Moreover, a trace-driven simulation on larger distributed clusters indicates the good scalability of our QoS-aware strategy.

Note that, our scheduler is equally applicable in large-scale distributed systems such as datacenters and HPC systems.

An earlier conference version of this paper appeared in [27]. Here we extend the previous paper in several respects. Firstly, our benchmark is expanded from convolutional neural network (CNNs) to deep neural network (DNNs), including diverse CNNs and recurrent neural network (RNNs) models. We also provide a more exhaustive evaluation of DL workloads using multiple application configurations and diverse GPU localities. From our characterization, more valuable observations are drawn to support the scheduling design. Secondly, a complete system modeling and problem statement are included to show the details of our scheduling framework. Thirdly, additional experiments, numerical statistics and analyses are also provided.

2 Background

DL, especially DNN, is a type of representation learning that automatically infers features from raw data in order to accomplish AI tasks. Diverse DNN models, such as CNNs and RNNs, have many similarities. The structures of DNNs are usually defined by a set of connections between different groups of neurons that perform the same function, known as layers. A well-trained DNN model can produce high-quality features for input images or speeches. The DNN computation includes training and inference. Training is a

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

process that uses a learning algorithm, such as stochastic gradient descent (SGD) and labeled training data, to tune the network parameters for certain applications. An inference is a process that deploys the trained model to test on another unlabeled dataset, which is done with the aim of evaluating the generalization of the trained model. Considering the complexity of DNNs and the large size of the datasets, the training and batch inferences are usually carried out in an iterative fashion. Moreover, DNN training and batch inference operate on a few samples of data simultaneously, known as a **mini-batch**. In each iteration, the DNN models process a batch of data by computing the billions of floating point operations. Essentially, training and batch inference are both iterative processes. And other task types that have the same characteristics as mini-batch processing and iterative processing can be added to our benchmarks.

With the increasing popularity of DL, numerous DL frameworks (including Tensorflow, MXnet and Caffe) have been proposed to optimize the various aspects of training and inference. Furthermore, most models in Model Zoo are fine-tuned using several typical DNN models, e.g., *Inception* [28] for image classification, and *Seq2seq* [29] for machine translation. These classic models are adopted in this work. For parallel or distributed systems, data parallelism and model parallelism are the primary sources of parallel tasks on multiple GPUs [30]. In terms of data parallelism, each device is responsible for the complete computing procedure of a DNN model. When each device completes the gradient calculation in each iteration, the gradients must be aggregated and used to update the parameters of the model. The aggregation leads to a significant communication overhead known as a **synchronous communication overhead**. In contrast, in model parallelism, an individual network's computation is distributed across multiple devices. The communication overhead results not only from parameter aggregation, but also from intermediate results communication across connections between devices. Due to the more complex communication requirement of model parallelism, placement design based on data parallelism is common in industry and academia, which is also the focus of this work. Meanwhile, in order to simplify the scheduling problem, we assume that each task always has **symmetric placements** on multiple devices. Each device has its own worker and parameter server to process the same scale subtask. Moreover, each device needs to communicate with others during synchronization. In recent years, although some new accelerators (such as FPGA, ASIC and TPU) have emerged, GPU still dominates in DL-related research. The comprehensive characterization and scheduling research of representative DL tasks on a GPU cluster are presented in the following section.

3 GENIE Overview

In this paper, we focus on how to exploit the GPU cluster to efficiently process diverse DL tasks in DL R&D scenarios. More specifically, given a GPU cluster platform $\mathcal{P}$ and a DL task queue $\Gamma = \langle \mathbf{t}_1, \mathbf{t}_2, ... \rangle$, our scheduling framework GENIE aims to identify the most appropriate placement for each task and the order of the task queue, achieving high users' QoS guarantee and system utilization. As shown in Fig. 2, GENIE contains two parts: offline profiling and online scheduling.

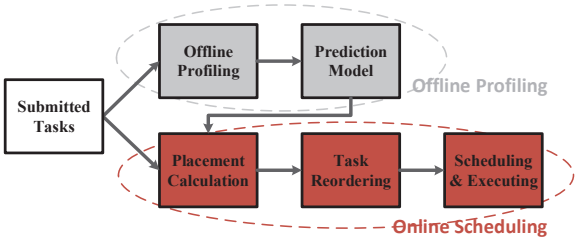


Fig. 2. The outline of our proposed scheduling framework GENIE. GENIE contains two parts: the offline profiling and the online scheduling. The offline part provides a prediction model based on the characterization of diverse DL workloads. The online part exploits the prediction model and users' QoS to design the scheduling strategy, including task placements and executing orders.

3.1 Offline Profiling

Offline profiling constructs a prediction model for diverse DL workloads, supporting the following scheduling design. The prediction model is based on **system modeling**, which in this paper includes platform modeling and task modeling. For a specific GPU cluster, platform modeling $\mathcal{P}$ is related to the total number of GPUs and a hierarchical GPU connection. Moreover, in a task queue Γ, DL tasks have diverse application configurations and QoS requirements. The challenge of modeling tasks is to find the key factors among the application configurations and task placements that have a significant impact on processing performance. In order to present practical DL task models and platform models, Section 4 proposes a comprehensive characterization for DL tasks under diverse configurations and placements on a GPU cluster. Based on the platform modeling and task modeling, the offline part presents the prediction model by means of a lightweight profiler. More details about the offline part are described in Section 5.1.

3.2 Online Scheduling

Based on the prediction model from the offline part, the online part in Fig.2 proposes a dynamic QoS-aware scheduling strategy. In the online part, our strategy decides the task placement and executing order for each task according to the principle of **shortest waiting allowance first (SWAF)**. In order to dynamically achieve high users' QoS guarantee and system utilization, our QoS-aware scheduling strategy is implemented based on an event-driven mechanism and makes dynamic scheduling decisions on specific time points. We will give more details about the online scheduling part in Section 5.

4 DL Workload Characterization

In this section, a comprehensive characterization is proposed to qualitatively evaluate diverse DL models in order to construct the practical task models. The factors affecting task performance include the application configurations and the task placements. We select several typical CNN models (*Inception-v3* and *ResNet-50*) and RNN models (*Regularized*

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

TABLE 1
Platform Specifications

	Specifications
Node	Powerleader 4U PR4712GW Chassis * 4
System	CentOS 7.0, Tensorflow 1.7.0
CPU	Intel Xeon E5-2660 v3, 10C, 2.60GHz * 2 / node
Memory	64GB DDR3 Memory / node
GPU	NVIDIA Tesla K80 * 4 / node
Interconnection	56Gbps InfiniBand, PCIe 3.0 x16 in node, no NVlink

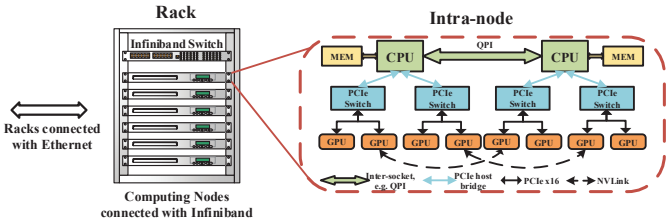


Fig. 3. Inter- and intra-node Topology for GPU clusters. Nodes within the same rack typically have InfiniBand links, while cross-rack traffic goes through the ethernet. Moreover, there is a hierarchy of network connectivity among GPUs in a node.

LSTM and *Seq2Seq*) and evaluate them on a 16-GPU cluster. The specifications of the 16-GPU cluster, which is adopted as our DL R&D platform, are shown in Table 1. Considering that Tensorflow has numerous users for application development, in this paper, we adopt Tensorflow 1.7.0 as our implementation framework.

Our evaluation is focused on the impacts of diverse application configurations and task placements. Based on the characterization, we analyze and conclude on the observations to guide the subsequent task modeling and scheduling. Due to space limitations, we show the results of *Inception-v3* and *Seq2seq*. Other DL models have the same features and properties.

4.1 The Impacts of Application Configurations

In DL research and development scenarios, users and developers typically try several application configurations, called hyper-parameters, of a specific model to identify the qualified configurations. Such conditional exploration, called a hyper-parameter search, can be manual or automated. Apart from the model structure, the hyper-parameters include the learning rate, step size, batch size, the number of iterations and momentum, which affect convergence, the level of parallelism, accuracy, and the processing rate. However, in this section, we only focus on the response latency and processing rate from the view of the system, without a change on model accuracy and convergence. The processing rate is measured by *samples (images or words)/second*. Based on previous studies [6, 22] and numerous experiments, we observe and analyze that **batch size** and **the number of iterations** have significant effects on these metrics.

Observation 1*: For DL tasks, the response latency is linearly correlated with the number of iterations. The processing rate increases at first and then plateaus as the batch size continues to increase on a single GPU.*

As shown in Fig. 4, we present the evaluation of *Inception-v3* and *Seq2seq* to show the impacts of performance with varying application configurations **on a single GPU**. More specifically, the results show that the latency increases linearly with the number of iterations for various DL models. With a given number of iterations, it is easy to predict the response latency. In contrast, batch size has more complex impacts on the processing rate for diverse DL tasks, as illustrated in Fig. 4(b). As the batch size continues to increase, the processing rate also increases, sharply at first, and it then plateaus when the batch size reaches a certain point. The reason is that the GPU computing resources become saturated until the batch size reaches a certain point. Compared to training, inference has less computation and a higher processing rate.

Observation 2*: GPU utilization is positively correlated with batch size. Compared with training, inference has lower GPU utilization. The GPU memory occupancy depends on the model structures and DL frameworks.*

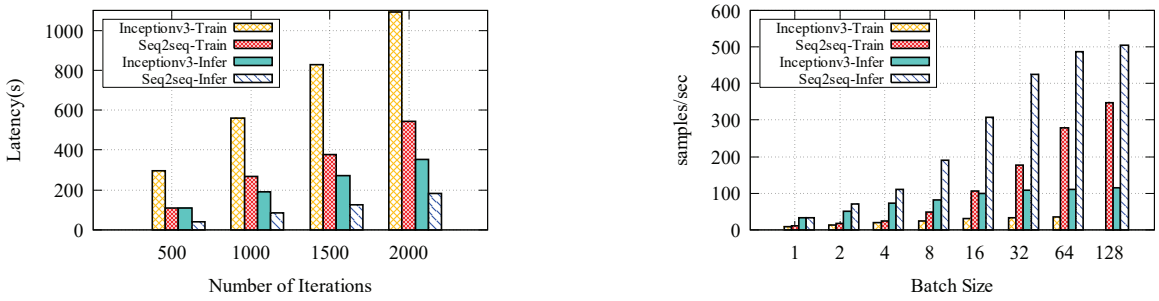
Meanwhile, we give a detailed profiling for diverse DL tasks using NVIDIA Profile to conduct a performance analysis. The metrics we have adopted include GPU utilization and GPU memory occupancy, which are both affected by batch size. As shown in Fig. 5(a), GPU utilization is positively correlated with batch size. However, for inference tasks, GPU utilization plateaus when the batch size reaches a certain point. This occurs because the fast inference process cannot hide the read latency of the input images in each iteration. Moreover, GPU memory occupancy is another key factor. By default, Tensorflow pre-allocates nearly all GPU memory of all GPUs visible to the single process [31], which cannot support multiple tasks. To change this, we set the **allow_growth** option in Tensorflow, which attempts to allocate GPU memory as usage grows. The maximum GPU memory occupancies with varying batch sizes are shown in Fig. 5(b). Obviously, apart from batch size, the GPU memory occupancy is dominated by the Tensorflow framework, which leads to irregular increases in the memory occupancy. Considering the dynamic and unpredictable GPU occupancy, GPUs are considered exclusive and not shared by multiple tasks simultaneously. The exclusive mechanism can make the following modeling and scheduling design more concise.

4.2 The Impacts of Diverse Placement Strategies

Generally, task placement is related to the platform model. As mentioned in Section 3.1, the key factors of the platform model for GPU clusters include the total number of GPUs and the hierarchy GPU connection. However, compared with datacenters and HPC systems, GPU clusters adopted as DL R&D platforms have a similar cluster setup but a limited scale. As shown in Fig. 3, nodes within the same rack typically have InfiniBand links, while cross-rack traffic goes through the ethernet. Moreover, there is a hierarchy of network connectivity among GPUs in a node. Considering the actual situations of and cost implications for most EIUs, in this paper, our DL R&D platform is limited in a rack, without cross-rack traffic and NVLink interconnects in a node. Moreover, in order to simplify the problem, we assume that the GPU cluster in our paper is **symmetric**

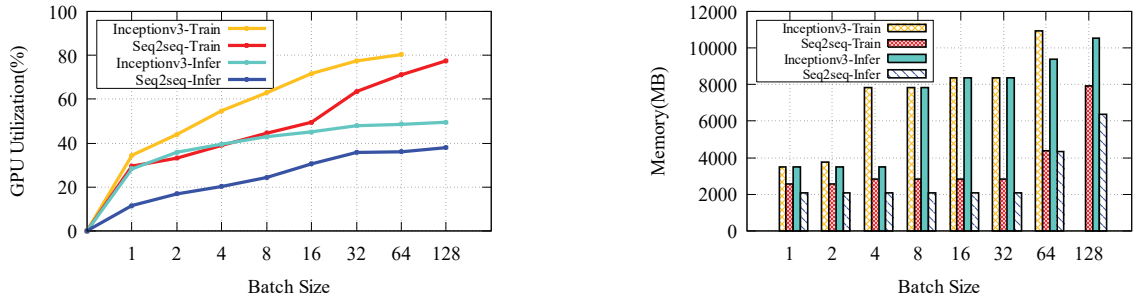
Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558



(a) Latency with the number of iterations (b) Processing rate with batch sizes

Fig. 4. Impact of performance with varying application configurations on a single GPU. Limited by the GPU memory, *Inception-v3* model cannot be trained with batch size 128 on a single GPU. The *samples/sec* of processing rate is *image/s* for *Inception-v3* and *word/s* for *Seq2seq*.



(a) GPU Utilization (b) GPU Memory Occupancy

Fig. 5. GPU resource occupancy with varying batch sizes on a single GPU.

and that each node has the same number of CPUs and GPUs. Therefore, the platform models $\mathcal{P}$ in our paper are based on the number of nodes, the number of GPUs per node, and the hierarchical GPU connection. Moreover, we denote **the number of nodes** and **the number of GPUs in each node** to indicate diverse task placements, which cannot exceed the upper limits of our platform. The impacts of the hierarchy GPU connection are reflected in our evaluation of GPU scalability and locality.

Moreover, in this paper, we focus on data parallelism and symmetric placement. Based on data parallelism, for a DL task with a fixed global batch size, the more nodes and GPUs we use, the smaller the local batch size on each GPU is. This distributed training method on multiple devices has been proven to obtain an equivalent training effect compared to training on a single GPU [32]. The symmetry placement determines that each sub-task divided on each GPU has the same computation. We then propose the evaluation of the impacts under diverse placements for DL tasks.

***Observation 3**: The processing rate of a DL task is determined by the processing rate of each GPU and the synchronization overhead among GPUs. The task with a larger batch size is preferred for spreading across multiple GPUs, while the task with a smaller batch size is preferred for packing on less GPUs or a single GPU. Inference tasks have linear scaling with the increasing total number of GPUs due to the absence of synchronization.*

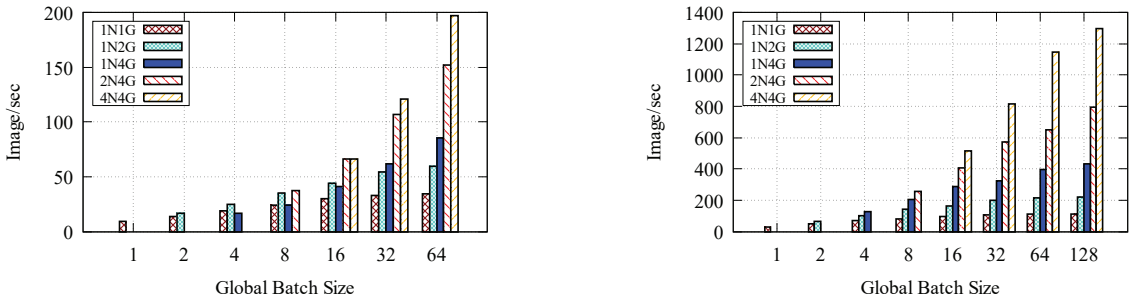
Taking *Inception-v3* as an example, we first present the GPU scalability results in Fig. 6 and the impacts thereof with an increasing total number of GPUs and varying batch sizes. When the number of GPUs exceeds 4, the task would be spread over multiple nodes. As shown in Fig. 6(a), for varying global batch sizes, the overall processing rates have different trends as the total number of GPUs increases in one node or multiple nodes. When the global batch size is small, the processing rate has no significant improvements as more GPUs are adopted (when the global batch size is smaller than 16). This is because the model synchronization overhead among GPUs dominate in response latency, and GPU utilization is low due to the small local batch size assigned to each GPU. When the global batch size is increasing in size, the training performance increases sharply along with the increasing number of GPUs (when the global batch size is larger than 16). This is because the single GPU's utilization is becoming saturated and the synchronization frequency and overhead percentages among GPUs decrease. However, spreading over multiple GPUs may lead to resource fragmentation [20]. For inference tasks, as shown in Fig. 6(b), the processing rate achieves almost linear scaling along with the increasing total number of GPUs due to the absence of synchronization among GPUs.

***Observation 4**: The DL training tasks with smaller batch sizes are more sensitive to the GPU locality than those with larger batch sizes.*

We evaluate the different levels of sensitivity to GPU locality, that are correlated to synchronization frequency and overheads. Considering that inference tasks have no synchronization during runtime, they also have no sensitivity to GPU locality. Due to space limitations, we only present here the evaluation results of training tasks. Fig. 7(a) shows the intra-node locality for *Inception-v3* training. When the model is trained with two GPUs in a node, there are three kinds of

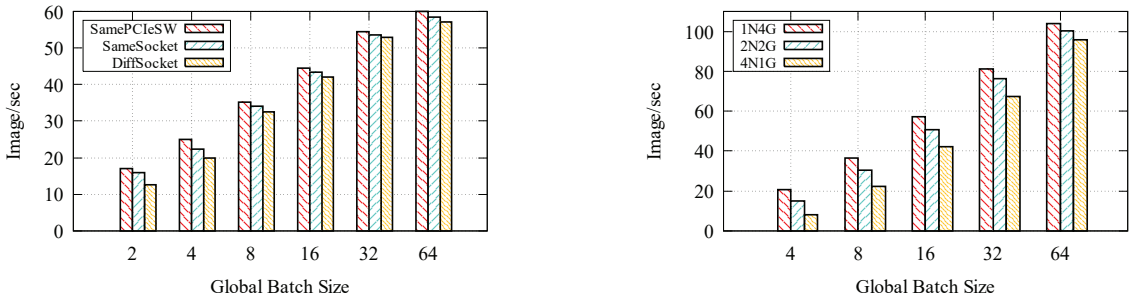
Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558



(a) Training (b) Inference

Fig. 6. GPU scalability. For diverse placements, *N* and *G* represent the number of nodes and the number of GPUs per node respectively. The total number of GPUs is increasing from *1N1G* to *4N4G*.



(a) Intra-node (b) Inter-node

Fig. 7. (a) Intra-node locality for GPU training. *SamePCIeSW*, *SameSocket* and *DiffSocket* denote two GPUs on the same PCIe Switch, two GPUs on the different PCIe Switch but on the same CPU socket, and two GPUs on the different CPU socket, respectively. (b) Inter-node Locality for GPU Training.

GPU localities: two GPUs on the same PCIe Switch (denoted as *SamePCIeSW*); two GPUs on the different PCIe Switch but on the same CPU socket (denoted as *SameSocket*); and two GPUs on different CPU sockets (denoted as *DiffSocket*). As shown in Fig. 7(a), poor locality can lead to performance degeneration due to the model synchronization overhead in each mini-batch. More specifically, the degeneration worsens when the global batch size decreases. When the batch size is increasing, the synchronization frequency decreases, and the communication load on the underlying PCIe bus is alleviated [6]. Similar trends are observed on multiple nodes, as shown in Fig. 7(b). When *Inception-v3* is trained with four GPUs, there are three types of inter-node localities: four GPUs in a node (denoted as *1N4G*), four GPUs in two nodes (denoted as *2N2G*), and four GPUs in four nodes (denoted as *4N1G*). Obviously, the communication overhead among nodes is larger than that among GPUs in one node.

5 Scheduling Policy

Based on the qualitative analysis in the previous section, in this section, we focus on system modeling and propose our QoS-aware scheduling framework GENIE for DL R&D scenarios. As shown in Fig.2, we explain the details of offline profiling and online scheduling. Section 5.1 presents system modeling and a prediction model for a single DL task under diverse configurations and placements. The prediction model is based on offline profiling results and is conducted by a lightweight profiler. According to the prediction model, Sections 5.2-5.4 present our complete QoS-aware scheduling strategy to dynamically determine task placements and executing order, achieving high users' QoS guarantee and system utilization. Complete notation concerning the terms used for modeling and scheduling is shown in Table 2.

5.1 Prediction Modeling for Single DL Task

5.1.1 Platform Modeling and Task Modeling

Firstly, we present our system modeling, including platform models and task models. According to the platform modeling explained in Section 4.2, a given homogeneous platform is denoted as a triple

$$\mathcal{P} = \langle N, G, Comm(\cdot)\rangle, \tag{1}$$

where N and G represent the upper limits of the number of nodes and the number of GPUs per node respectively. Meanwhile, $Comm(\cdot)$ refers to a communication penalty function for prediction modeling. The communication penalty is related to the hierarchical GPU connection of platform $\mathcal{P}$.

According to the characterization of Section 4.1, the key factors, including task type, batch size and the number of iterations, are selected from diverse application configurations to model tasks. Therefore, a given task is denoted as a triple

$$\mathbf{t} = \langle w_{\text{type}}, w_{\text{bat}}, w_{\text{iter}}\rangle, \tag{2}$$

where w_{type}, w_{bat} and w_{iter} represent the task type, global batch size and number of iterations respectively.

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

TABLE 2
Notation used for modeling and scheduling

Notation	Description
$\mathcal{P} = \langle N, G, Comm(\cdot)\rangle$	A DL R&D platform, where N, G and $Comm(\cdot)$ reprensent the upper limits of the number of nodes, the number of GPUs per node and the communication penalty function.
$\mathbf{t} = \langle w_{\text{type}}, w_{\text{bat}}, w_{\text{iter}}\rangle$	A DL task, where w_{type},w_{bat},w_{iter} represent the task type, global batch size and the number of iterations.
$\mathbf{s} = \langle d_{\text{node}}, d_{\text{g/n}}\rangle$	A placement of task $\mathbf{t}$, where d_{node} and $d_{\text{g/n}}$ represent the number of nodes and the number of GPUs per node.
$Lat(\mathbf{t}, \mathbf{s})$	The response latency of task $\mathbf{t}$ under placement $\mathbf{s}$.
$PR(\mathbf{t}, \mathbf{s})$	The processing rate of task $\mathbf{t}$ under placement $\mathbf{s}$. $sPR(\mathbf{t}')$ represents the processing rate for task $\mathbf{t}'$ on a single GPU.
$Comm(\mathbf{s})$	The synchronous communication overhead under placement $\mathbf{s}$.
$Cost(\mathbf{s})$	The cost of placement $\mathbf{s}$.
$CER(\mathbf{t}, \mathbf{s})$	The cost-effective ratio of task $\mathbf{t}$ under placement $\mathbf{s}$.

More specifically, task type includes the model type, e.g., *Inception-v3*, and the calculation type, i.e., *training* and *inference*. Moreover, task placements have significant impacts on processing rate and response latency. According to *Observation 4*, considering that the intra-node GPU locality has less of an impact on performance, we prefer choosing the placements with the best available GPU locality in one node. Therefore, the placement on a GPU cluster for a task is denoted by a tuple

$$\mathbf{s} = \langle d_{\text{node}}, d_{\text{g/n}}\rangle, \tag{3}$$

including the number of nodes d_{node} and the number of GPUs per node $d_{\text{g/n}}$. Based on the symmetric placement hypothesis mentioned in Section 2, a task is equally distributed across multiple nodes and GPUs.

5.1.2 Performance Prediction Model

A prediction model for a single DL task forms the basis of the following multi-task scheduling design. According to *Observation 2*, GPUs are adopted as exclusive accelerators in this paper due to unpredictable GPU occupancy. Therefore, the interference of GPU sharing by multiple tasks can be ignored. Diverse DL tasks, including training and batch inference, are iterative in nature. The response latency depends on the whole computation and processing rate. The response latency of task $\mathbf{t}$ under placement $\mathbf{s}$ is calculated as

$$Lat(\mathbf{t}, \mathbf{s}) = \frac{w_{\text{bat}} \cdot w_{\text{iter}}}{PR(\mathbf{t}, \mathbf{s})} + \nu, \tag{4}$$

where $PR(\mathbf{t}, \mathbf{s})$ represents the processing rate, and $w_{\text{bat}} \cdot w_{\text{iter}}$ represents the whole computation of the task. Moreover, ν represents the startup overhead of the GPUs. Generally, the startup overhead of the GPUs is related to task $\mathbf{t}$ and the occupied number of GPUs. However, in Tensorflow, all GPUs are started by default, then the startup overhead is degenerated to a constant value.

According to our *Observation 3* in Section 4, the overall processing rate of a task under a distributed placement is determined by **the processing rate of each single GPU** and **the performance reduction caused by the synchronous communication overhead among GPUs**. Therefore, the processing rate of diverse DL tasks under varying application configurations and placements denoted by PR is denoted as

$$PR(\mathbf{t}, \mathbf{s}) = (d_{\text{node}} \cdot d_{\text{g/n}} - Comm(\mathbf{s})) \cdot sPR(\mathbf{t}'), \tag{5}$$

where $sPR(\mathbf{t}')$ and $Comm(\mathbf{s})$ are the processing rate of each single GPU and the communication penalty function respectively. Meanwhile, $\mathbf{t}'$ is the subtask assigned to each occupied GPU. They are discussed in Section 5.1.3 and Section 5.1.4. In Eq. 5, $d_{\text{node}} \cdot d_{\text{g/n}}$ is the total occupied number of GPUs under placement $\mathbf{s}$. Ideally, the overall processing rate is determined by the product of the total number of GPUs and the processing rate of each single GPU, assuming there is no communication overhead. However, the synchronous communication overhead among GPUs introduces an additional performance reduction, denoted as $Comm(\mathbf{s})$.

5.1.3 Processing Rate of a Single GPU

Task $\mathbf{t}$ under placement $\mathbf{s}$ is equally divided across multiple nodes and GPUs based on the global batch size. Each subtask assigned to each GPU has the same local batch size. That is, subtask $\mathbf{t}'$ is denoted as

$$\mathbf{t}' = \langle w_{\text{type}}, w'_{\text{bat}}, w_{\text{iter}}\rangle = \langle w_{\text{type}}, \frac{w_{\text{bat}}}{d_{\text{node}} \cdot d_{\text{g/n}}}, w_{\text{iter}}\rangle, \tag{6}$$

where w'_{bat} is the local batch size assigned to each single GPU. According to *Observation 1* in Section 4, we observe a positive correlation between the processing rate of each single GPU sPR and the **local batch size** w'_{bat}. Based on the correlation, we adopt a polynomial function to model the processing rate of a single GPU. Therefore, the processing rate of a single GPU sPR is described as

$$sPR(\mathbf{t}') = \sum_i k_i (w'_{\text{bat}})^i = \sum_i k_i (\frac{w_{\text{bat}}}{d_{\text{node}} \cdot d_{\text{g/n}}})^i, \tag{7}$$

where k_i is the coefficient depending on task type w_{type} and the platform model $\mathcal{P}$. In order to simplify the modeling and avoid over-fitting, the degree of the polynomial regression is made as simple as possible. Our experiments show that a quadratic polynomial, i.e. $i = 2$, is sufficient for modeling the processing rate of each single GPU.

5.1.4 Synchronous Communication Overhead

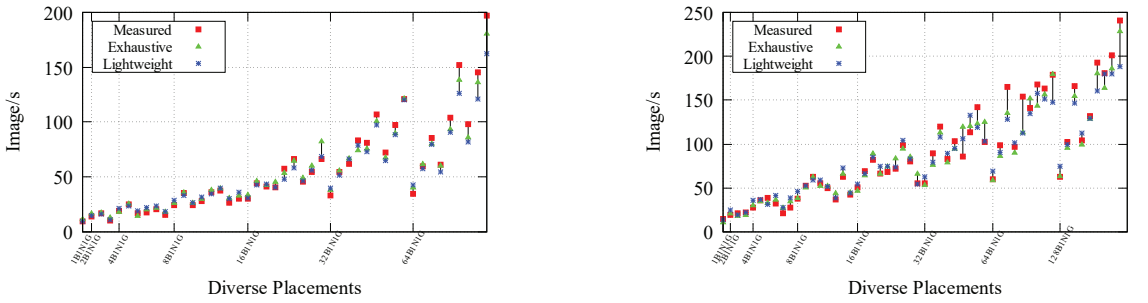
When the synchronous communication overhead among the nodes and GPUs is ignored, the ideal performance can be obtained by calculating the total number of GPUs and the performance of each single GPU. However, the synchronous communication overhead leads to additional performance breakdown. In this paper, we introduce a communication penalty function $Comm(\cdot)$, which is based on the **average path length among GPUs**, to quantify these overheads. The communication penalty function $Comm(\cdot)$ is described as

$$Comm(\mathbf{s}) = \begin{cases} 0 & d_{\text{node}}{=}d_{\text{g/n}}{=}1, \\ \frac{((d_{\text{node}}-1)\cdot d_{\text{g/n}} + \lambda(d_{\text{g/n}}-1))\cdot\gamma}{d_{\text{node}}\cdot d_{\text{g/n}}-1} & d_{\text{node}}, d_{\text{g/n}}{>}1. \end{cases} \tag{8}$$

As mentioned in Section 2, during the DL training synchronization, each GPU must communicate with each other.

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558



(a) Inception-v3 (b) ResNet50

Fig. 8. *Inception-v3* and *ResNet50* training performances under diverse placements on a 4-node, 16-GPU DL cluster. 'Measured', 'Exhaustive' and 'Lightweight' represent the real value and the theoretical values calculated by prediction models of exhaustive and lightweight profiling, respectively.

Therefore, the communication penalty function is determined by the topology of placement $\mathbf{s}$. Taking any one GPU in placement $\mathbf{s}$ as an example, $(d_{\text{node}} - 1) \cdot d_{\text{g/n}}$ and $(d_{\text{g/n}} - 1)$ represent the number of GPUs across different nodes and the number of GPUs in the same node for communication. The sum of the total number of GPUs for communication is $d_{\text{node}} \cdot d_{\text{g/n}} - 1$. Moreover, γ is a coefficient associated with the hierarchical GPU connection of platform $\mathcal{P}$, and λ is a balance factor that normalizes the different communication capabilities in the same node and across different nodes. The coefficient γ and the balance factor λ are obtained from the profiling and regression. In addition, when $d_{\text{node}} = d_{\text{g/n}} = 1$, the task is running on a single GPU and Eq. 5 degenerates to Eq. 7. For inference tasks, $Comm(\cdot)$ is equal to 0 due to the fact that there is no communication among iterative computings.

5.1.5 Lightweight Profiling Method

Clearly, there are several types of coefficients, k_i, γ and λ, that are related to task type w_{type} and the experimental platform $\mathcal{P}$. They are derived by unary or multiple regression according to the profiling results. For a given task type w_{type} and platform $\mathcal{P}$, we assume that there are B, N and G different configurations for the global batch size, the number of nodes, and the number of GPUs per node on a DL GPU cluster. During offline profiling, the overall experimental cost of the profiling is $B \times N \times G$ in terms of obtaining the prediction model for task type w_{type} on platform $\mathcal{P}$. However, we observe that this exhaustive profiling method is too costly, and there is much redundant information. In order to decrease the profiling cost, we conduct **a lightweight profiling method** that can obtain sufficient results for a coefficient regression.

Firstly, Eq. 7 shows that the processing rate is only related to the batch size on a single GPU. Therefore, the regression of k_i only requires the experiments to be undertaken B times on a single GPU to obtain the relationship between the batch size and the processing rate. Likewise, the regression of γ and λ requires $N + G$ runs for a fixed global batch size. Specifically, G times experiments are used for evaluating the impacts of the number of GPUs per node, and N times experiments are used for evaluating the impacts of the number of nodes respectively.

A training performance comparison is shown in Fig. 8. The average prediction error conducted by our lightweight profiler for diverse DL models is less than 5%. Although more profiling data can be used for finetuning to achieve higher prediction accuracy, our lightweight profiler has achieved comparable accuracy with limited experiments, which is sufficient for scheduling design. Greater evaluations of prediction accuracy in multi-task scheduling scenarios are presented in Section 7. Meanwhile, the lightweight profiler is highly efficient, as $B + N + G \ll B \times N \times G$. For example, a DL task type have $10, 15, 16$ configurations for B, N and G. The total number of lightweight profiling experiments is almost 40 times. Due to the iterative fashion of DL task, each experiment only needs approximately 100 iterations, which costs average 0.5 minute. Each type of DL model only needs to be profiled once. More details on the offline profiling can refer to Section 6. Therefore, the modeling and the scheduling framework have good generalization and scalability by means of the lightweight profiler on diverse types of models and larger scale of clusters, including datacenters and HPC systems.

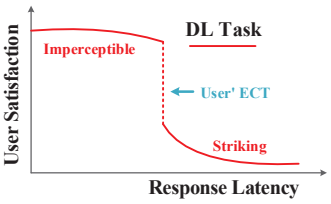


Fig. 9. User satisfaction with varying response latency

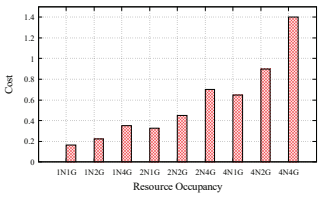


Fig. 10. Cost model ($\theta = 0.4$)

5.2 User-Oriented and System-Oriented Metrics

Before describing the details of our QoS-aware scheduling algorithm, we propose some user-oriented and system-oriented metrics in this paper, which are conducive to the following problem statement and scheduling design. Section 5.2.1 presents our QoS definition from the user's perspective, and Section 5.2.2 provides a cost-effective model from a system perspective.

5.2.1 Definition of User's QoS

Firstly, we consider the QoS metrics in this paper. For a multi-user DL R&D platform, the expected completion time (ECT) e is a key indicator of the QoS of the DL task. In

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

most cases, it is acceptable for users that the submitted tasks are finished before ECT [33]. As shown in Fig. 9, if the response latency exceeds the ECT, users' QoS satisfaction would decrease sharply. Considering that ECT is related to the computation of the task, we refer to [34] and denote the double baseline response latency of the task execution on a single GPU as the normal users' ECT. In emergencies of varying degrees, users can apply for higher priorities to shorten their ECTs. For task $\mathbf{t}$, we define three classes of user priorities and their associated ECTs e as

$$e = \begin{cases} 0 & \text{Urgent,} \\ Lat(\mathbf{t}, \langle 1,1 \rangle) & \text{Prior,} \\ 2Lat(\mathbf{t}, \langle 1,1 \rangle) & \text{Normal,} \end{cases} \tag{9}$$

where $\langle 1,1 \rangle$ represents the placement on a single GPU and $d_{\text{node}} = d_{\text{g/n}} = 1$. Note that the ECTs of the *Urgent* tasks are zero, meaning that these tasks have the highest priorities and should be finished as soon as possible. However, such tasks have to violate their QoS regardless of the placements chosen. In the following evaluation, we adopt the QoS-guarantee percentage to show the average QoS satisfaction of users.

5.2.2 A Cost-Effective Model with Diverse Placements

Based on *Observation 3*, a higher allocation occupancy of computing resources can lead to performance improvement in most cases. Moreover, some tasks are less sensitive to GPU locality, especially for tasks with large batch sizes. In a multi-task scenario, cross-node placement is usually unavoidable. For these tasks, a spread placement across multiple nodes can also achieve similar performance than a packed placement in one node from *Observation 4*. Therefore, a scheduling framework should consider diverse placements to improve the efficiency of resource allocations in order to avoid resource fragmentation [20]. We introduce cost model $Cost(\cdot)$, which is determined by the total number of GPUs and the topology among them, to estimate resource occupancy and fragmentation. The cost model is denoted as

$$Cost(\mathbf{s}) = \frac{d_{\text{node}} \cdot d_{\text{g/n}}}{N \cdot G} + \theta \frac{d_{\text{node}}}{N}, \tag{10}$$

where N and G represent the upper limits of d_{node} and $d_{\text{g/n}}$ provided by the cluster. In Eq. 10, the first item represents the normalized total GPU occupancy. The second item reflects the impacts on the topology of the total occupied GPUs and θ is the weight factor that is used to adjust the balance between two items. In our definition of a cost model, the more concentrated the occupied GPUs are, the lower the cost is. An example of the cost model is shown in Fig. 10. Based on Eq. 5 and Eq. 10, the cost-effective ratio (CER) can be calculated as follows:

$$CER(\mathbf{t}, \mathbf{s}) = \frac{PR(\mathbf{t}, \mathbf{s})}{Cost(\mathbf{s})}, \tag{11}$$

which is significant for the following scheduling design to achieve high system utilization.

5.3 Problem Statement and Scheduling Objective

We present the complete problem statement here. Given the platform $\mathcal{P}$ with the communication penalty function $Comm(\cdot)$ and the upper limits of the number of nodes N and the number of GPUs per node G, there is a task queue $\Gamma = \langle \mathbf{t}_1, \mathbf{t}_2, ... \rangle$. The ECTs of the tasks are $e_1, e_2, ...$ respectively. For each task $\mathbf{t}_i$, we make a trade-off between GPU scalability and resource fragmentation to search for an optimal placement $\mathbf{s}_{i*}$ that can guarantee the ECT of $\mathbf{t}_i$ with the highest CER as best effort. Then, the executing order $\mathbf{O}$ of the tasks must be dynamically adjusted for users' QoS considering the emergencies of varying degrees. In summary, considering the limited computing resources and QoS constraints, the aim of our scheduling strategy is to **guarantee users' QoS while improving system utilization as a best-effort delivery** by determining the scheduling orders and placements of tasks. The QoS-guarantee percentage $\mathbb{Q}$, the percentage of the tasks that do not exceed ECTs, and makespan $\mathbb{M}$ are denoted as the scheduling objectives. The scheduling problem in our paper can be formalized as

$$\begin{aligned} &\text{Given: } \mathcal{P} = \langle N, G, Comm(\cdot) \rangle \text{ and } \Gamma = \langle \mathbf{t}_1, \mathbf{t}_2, ... \rangle, \\ &\text{find: } \mathbf{S} = \langle \mathbf{s}_{1*}, \mathbf{s}_{2*}, ... \rangle \text{ and } \mathbf{O} = \langle \mathbf{t}'_1, \mathbf{t}'_2, ... \rangle, \\ &\text{maximizing: } \mathbb{Q}, \text{ minimizing: } \mathbb{M}, \\ &\text{subjected to: } \mathbf{s}_{i*} = \langle d^{(i)}_{\text{node}}, d^{(i)}_{\text{g/n}} \rangle, i \in [1, 2, ...], \\ &\qquad d^{(i)}_{\text{node}} < N, d^{(i)}_{\text{g/n}} < G. \end{aligned}$$

5.4 A QoS-aware Scheduling Algorithm

Based on the problem statement, we propose a QoS-aware dynamic scheduling algorithm on a GPU cluster. For a given platform $\mathcal{P}$, the task queue keeps changing online throughout runtime with old tasks finishing and new tasks arriving constantly. In order to dynamically achieve the above scheduling objectives, our QoS-aware scheduling algorithm is implemented based on an event-driven mechanism, and we choose the specific time points to make dynamic scheduling decisions. The scheduling time points are derived from the events of task completion or a new arrival. We then describe a snapshot of our QoS-aware scheduling process at a specific time point.

At each event-based time point, we propose a scheduling algorithm based on the principle of **shortest waiting allowance first (SWAF)** to make the current optimal scheduling decision. Assuming that the current time is t_{curr}, the tasks that arrive at time t_{curr} are pushed into a waiting queue $\mathbf{Q}_{\text{curr}}$. $\mathbf{Q}_{\text{curr}}$ has tasks

$$\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_n, \ n = |\mathbf{Q}_{\text{curr}}|$$

whose ECTs are, respectively,

$$e_1, e_2, ..., e_n.$$

For each task, we calculate the response latency l and CER r of each placement according to Eq. 4 and Eq. 11. For example, there are q_i placements

$$\mathbf{s}_{i1}, \mathbf{s}_{i2}, ...\mathbf{s}_{iq_i}, \ q_i = |\mathbf{M}_i|$$

in a placement set $\mathbf{M}_i$ for task $\mathbf{t}_i$, whose response latencies and CERs are, respectively,

$$\begin{aligned} &l_{i1}, l_{i2}, ...l_{iq_i}, \ q_i = |\mathbf{M}_i| \\ &r_{i1}, r_{i2}, ...r_{iq_i}, \ i \in [1, 2, ..., n]. \end{aligned}$$

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

Algorithm 1: A Snapshot of QoS-aware Dynamic Scheduling Process on Time Point t_{curr}

Input: Waiting Queue $\mathbf{Q}_{\text{curr}}$, Current Time t_{curr}, Platform $\mathcal{P} = \langle N, G, Comm(\cdot)\rangle$
Output: Reordered Queue $\mathbf{Q}'_{\text{curr}}$

1 **for** each task $\mathbf{t}_i \in \mathbf{Q}_{curr}$ **do**
2 Obtain ECT e_i of task $\mathbf{t}_i$;
3 Generate all placements for task $\mathbf{t}_i$ under the limitations of platform $\mathcal{P}$ and store them in the set $\mathbf{M}_i$;
4 Initialize the palcement subset $\mathbf{M}'_i = \emptyset$;
5 **for** each placement $\mathbf{s}_{ij} \in \mathbf{M}_i$ **do**
6 Calculate response latency l_{ij} of task $\mathbf{t}_i$ under placement $\mathbf{s}_{ij}$ according to Eq. 4;
7 Calculate CER r_{ij} of task $\mathbf{t}_i$ under placement $\mathbf{s}_{ij}$ according to Eq. 11;
8 **if** placement $\mathbf{s}_{ij}$ satisfies ECT e_i according to Eq. 12 **then**
9 push placement $\mathbf{s}_{ij}$ into $\mathbf{M}'_i$;
10 **if** $\mathbf{M}'_i \neq \emptyset$ **then**
11 Select placement $\mathbf{s}_{i*}$ with the highest CER from $\mathbf{M}'_i$;
12 **else**
13 Select placement $\mathbf{s}_{i*}$ with the highest CER from $\mathbf{M}_i$;
14 Compute waiting allowance w_i according to Eq. 13;
15 Reorder $\mathbf{Q}_{\text{curr}}$ based on the shortest waiting allowance first (SWAF) schema, and obtain reordered queue $\mathbf{Q}'_{\text{curr}}$;
16 return $\mathbf{Q}'_{\text{curr}}$;

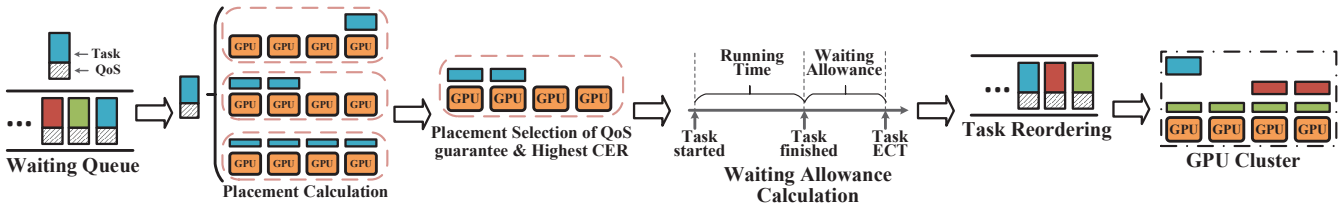


Fig. 11. The processing flow of our QoS-aware scheduling algorithm on time point t_{curr}. The blocks of different colors represent different DL tasks. The placement of each task is represented by the number of squares on the DL cluster. As tasks finish and new tasks arrive, the order and placement solution of each task in the waiting queue may keep changing.

In order to guarantee the user's QoS, we select the placements $\mathbf{s}_{ij} \in \mathbf{M}_i$, which can satisfy the QoS by

$$l_{ij} + t_{\text{curr}} \leq e_i,\ i \in [1, 2, ..., n],\ j \in [1, 2, ..., q_i]. \tag{12}$$

The placements $\mathbf{s}_{ij}$, which satisfy the above QoS requirements, are pushed into a subset of $\mathbf{M}_i$, denoted as $\mathbf{M}'_i$. The placement $\mathbf{s}_{i*}$ of the highest r among $\mathbf{M}'_i$ is adopted as the solution for task $\mathbf{t}_i$. A key indicator for current task priority, denoted as **waiting allowance** w, is calculated based on the placement $\mathbf{s}_{i*}$ by

$$w_i = e_i - (l_{i*} + t_{\text{curr}}),\ i \in [1, 2, ..., n],\ \mathbf{s}_{i*} \in \mathbf{M}'_i. \tag{13}$$

The order of all tasks is determined by the waiting allowances. The task with a smaller w in the task queue has a higher priority. Moreover, when there are no placements that can meet the user's ECT, the placement with the highest CER among $\mathbf{M}_i$ is adopted as the solution. Obviously, w could be negative, which has no impact on the task ordering. The scheduler then monitors the cluster load and launches the tasks according to the reordered waiting queue. When the cluster cannot meet the resource requirements, the tasks in the waiting queue keep waiting. As tasks finish and new tasks arrive, the order and placement solution of each task in the waiting queue may keep changing. A snapshot of our dynamic scheduling process is shown in Algorithm 1 and Fig. 11.

The number of the loops from *line 1* to *line 14* is $\sum_{i=1}^{n} q_i$. Meanwhile, other parts have constant computation overhead, including initialization, placement generation, latency and CER calculation, and placement selection. Therefore, the whole placement calculation (from *line 1* to *line 14*) has $O(\sum_{i=1}^{n} q_i) = O(n\bar{q})$ complexity. Moreover, the reordering (*line 15*), which is implemented by Heapsort, has $O(n \log n)$ complexity. The overall time complexity of our scheduling algorithm is $O(n\bar{q}) + O(n \log n) = O(n(\bar{q} + \log n))$. However, the average number of placements $\bar{q}$ on a limited scale GPU cluster is much less than the number of tasks n. According to $\bar{q} \ll n$, the time complexity of our QoS-aware algorithm can be deduced as $O(n \log n)$.

6 Implementation

We implemented a prototype of the scheduling framework, called GENIE, as a plugin for Tensorflow, prototyping our proposed QoS-aware scheduling strategy. As shown in Fig. 12, GENIE includes two stages: offline profiling and online scheduling. Offline profiling constructs a prediction model for diverse DL workloads. Based on the prediction model, online scheduling decides the task placement and executing order for each task.

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

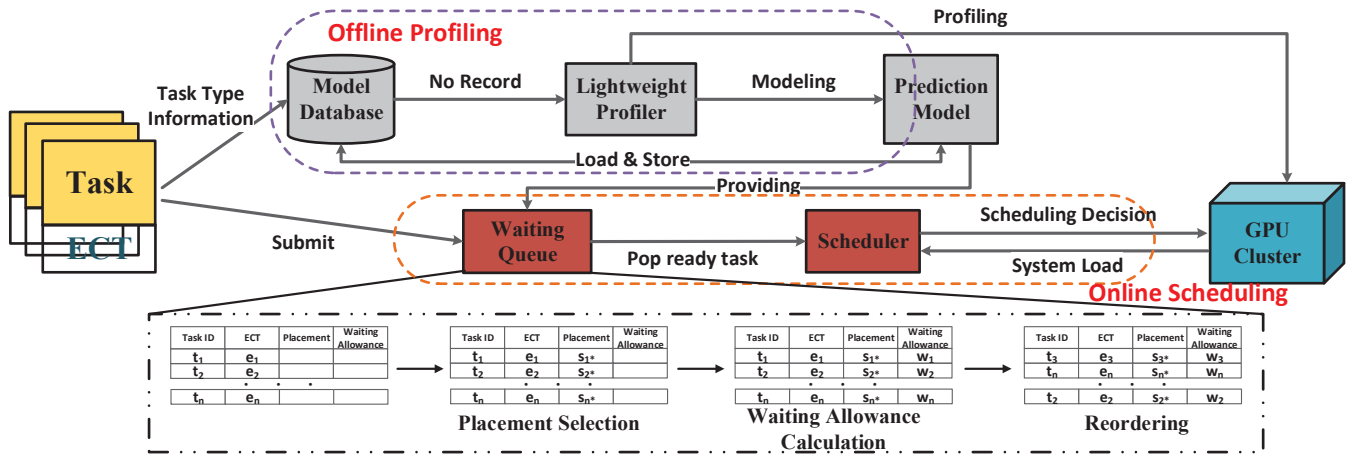


Fig. 12. The implementations of our scheduling framework GENIE. GENIE contains two parts: Offline Profiling and Online Scheduling. Each task is submitted to two parts simultaneously. For offline profiling part, the model database is in charge of storing and providing the prediction models for diverse task types. If there is no record in the model database, an existing prediction model is selected based on similarity comparison of model structures. Meanwhile, the lightweight profiler would make a profiling for the new task type when the GPU cluster has enough available resources and store the new prediction model into the model databased. For online scheduling part, the waiting queue keeps all waiting tasks. Based on the prediction model and an event-driven mechanism, the placement selection, waiting allowance calculation and reordering for all tasks are dynamically finished in the waiting queue. According to the system load, the ready task is popped from the queue to the scheduler and the scheduler launches the popped task on the GPU cluster.

Offline profiling part consists of a model database and a lightweight profiler. Before scheduling, we have made a profiling for most typical DNN models and stored the corresponding prediction models in the database. When each task is submitted, the task type information is retrieved from the model database. If the task type has been profiled before, the prediction model can be loaded from the database for subsequent scheduling. If there is no record, our framework would select an existing prediction model whose model structure has the highest similarity with the new task model. The similarity judgment is based on PB file and MetaGraph. When the GPU cluster has enough available resources, the lightweight profiler would make a profiling for the new task type and store the corresponding prediction model in the database.

Online scheduling part mainly includes a waiting queue and a scheduler. The arrived tasks are pushed into the waiting queue, where the selected placement and waiting allowance of each task are calculated according to the prediction models. Then the waiting queue is reordered according to the waiting allowances w. Moreover, the waiting queue is implemented according to a priority queue structure, an efficient implementation for task automatic ordering, and it is updated by an event-driven mechanism. The scheduler keeps monitoring the system load. When the cluster has enough idle resources, the task with the maximum priority is popped from the queue to the scheduler. The scheduler launches the popped task. Moreover, in this paper, GPUs are considered exclusive accelerators and tasks have private access to GPUs. Each task under a placement runs to completion without the GPU preemption mechanism.

7 Experimental Evaluation

In this section, we evaluate our proposed scheduling framework GENIE on a real GPU cluster. Furthermore, we provide a trace-driven simulation on larger-scale clusters to understand its scalability.

7.1 Experimental Setup

Platform: We adopt the GPU platform as shown in Table 1. Our platform consists of 4 nodes connected with a 56Gb InfiniBand. Each node has 2 CPUs (Intel Xeon E5-2660 v3 @ 2.60GHz) with 64GB of RAM and 4 GPUs (Nvidia Tesla K80) running CentOS 7.0 and Tensorflow 1.7.0. In order to gain more insight into the scalability of our proposed strategy, we have built a simulator that mimics various aspects of task logs, handling jobs with different arrivals as well as scheduling strategies.

Workloads: As there are no publicly available ML traces, we refer to [6, 21] and generate task queues containing diverse DL tasks with Poisson request arrivals. The specifications of the task generation are shown in Table 3. We adopt four classic CNN models, *Alexnet*, *GoogleNet*, *ResNet50*, and *Inception-v3*, and two classic RNN models, *Regularized LSTM* and *Seq2seq*, as the representative workloads in our evaluation. For each specific DL task, model type, calculation type, and application configurations (such as batch size and number of iterations) are generated with uniform distribution. According to the production workloads analysis [20], we set the QoS-priority proportions of *Urgent* and *Prior* users as 5% and 35%. Moreover, the duration of the task arrivals that we set here is 24 hours. In order to evaluate the scheduling strategies under various workload densities, we set the densities of multiple task queues as 5, 10 and 20 task arrivals per hour. Detailed evaluations of the parameter sensitivities are presented in the following section.

Baselines: The baselines, which are adopted for the purposes of comparison against our proposed QoS-aware scheduling algorithm, include the well-known scheduling policies in Yarn, Mesos, and Kubernetes.

- *FIFO Scheduler*: The *FIFO scheduler* simply allocates the available resources for the tasks in the order that they arrive in the task queue.

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

TABLE 3
Specifications of Task Queue Generation

	Type	Specifications
Model Type	CNN	*Alexnet*, *GoogleNet*, *ResNet50*, *Inception*-v3
	RNN	*Regularized LSTM*, *Seq2seq*
Calculation Type	Training	50%
	Inference	50%
QoS-Priority Proportion	Urgent	5%
	Prior	35%
	Normal	60%

- *Capacity Scheduler*: The *Capacity Scheduler* allocates the cluster resources to multiple types of models with capacity guarantees. Each model type can only access its own allocated resources.
- *Min-Min Scheduler[35]*: The tasks with earlier ECTs have higher priorities to be processed by *Min-Min Scheduler*.
- *Weighted Fair Scheduler[36]*: The weighted average value of the task's arrival time and ECT is adopted as ordering criterion. The tasks with smaller weighted values have higher priorities to be processed by *Weighted Fair Scheduler*.
- *Tetris[37]+Perf Scheduler*: *Tetris* usually allocates resources based on performance estimations. Since *Tetris* does not have its own mechanism to estimate a DL task, we adapt our prediction model to combine with *Tetris*. *Tetris+Perf Scheduler* adopts the resource allocation with the highest processing rate for each task.
- *Tetris+CER Scheduler*: Similarly, we adopt our cost-effective model to combine with *Tetris*. Considering both the prediction models and cost-effective models, the *Tetris+CER Scheduler* adopts the allocation solution with the highest CER for each task, without considering users' QoS.

From *Tetris+Perf Scheduler* to *Tetris+CER Scheduler* to our proposed QoS-aware strategy, the scheduling algorithms are incorporating more and more information and models for design. Compared with the above mentioned baselines, our proposed QoS-aware strategy takes users' QoS and cost-effective model into consideration for scheduling, achieving a trade-off between users' satisfaction and system utilization.

Metrics: As explained in Section 5.3, the motivation of this paper is to guarantee users' QoS and improve system utilization. Therefore, the metrics we adopt here are **makespan** and **QoS-guarantee percentage**. QoS-guarantee percentage $\mathbb{Q}$ is calculated as

$$\mathbb{Q} = \frac{NUM_{\text{comp}}}{NUM_{\text{task}}}, \tag{14}$$

where NUM_{comp} represents the number of tasks whose completion times do not exceed ECTs, and NUM_{task} represents the total number of the scheduled tasks. Moreover, considering that DL tasks have a wide spectrum of response latency, i.e., from a couple of hours to weeks, the **average normalization of response latency** [8], which can show the variation of response latency fairly, is also presented in our evaluation. The average normalization of response latency $\mathbb{L}$ for a task queue $\langle \mathbf{t}_1, \mathbf{t}_2, ... \rangle$ is calculated as

$$\mathbb{L} = avg(L_{\text{norm}}(\mathbf{t}_i)), i \in [1, 2, ...], \tag{15}$$

where $L_{\text{norm}}(\mathbf{t}_i)$ represents the normalization of the response latency for a single task $\mathbf{t}_i$. More specifically, $L_{\text{norm}}(\mathbf{t}_i)$ can be calculated as

$$L_{\text{norm}}(\mathbf{t}_i) = \frac{L_{\text{wall}}(\mathbf{t}_i)}{Lat(\mathbf{t}_i, \langle 1, 1 \rangle)} \tag{16}$$

where $L_{\text{wall}}(\mathbf{t}_i)$ represents the walltime of task $\mathbf{t}_i$ under a selected placement during runtime and $Lat(\mathbf{t}_i, \langle 1, 1 \rangle)$ represents the baseline response latency of the task execution on a single GPU, as defined in Eq. 4. Furthermore, the evaluation results presented later are the average of three runs for fairness.

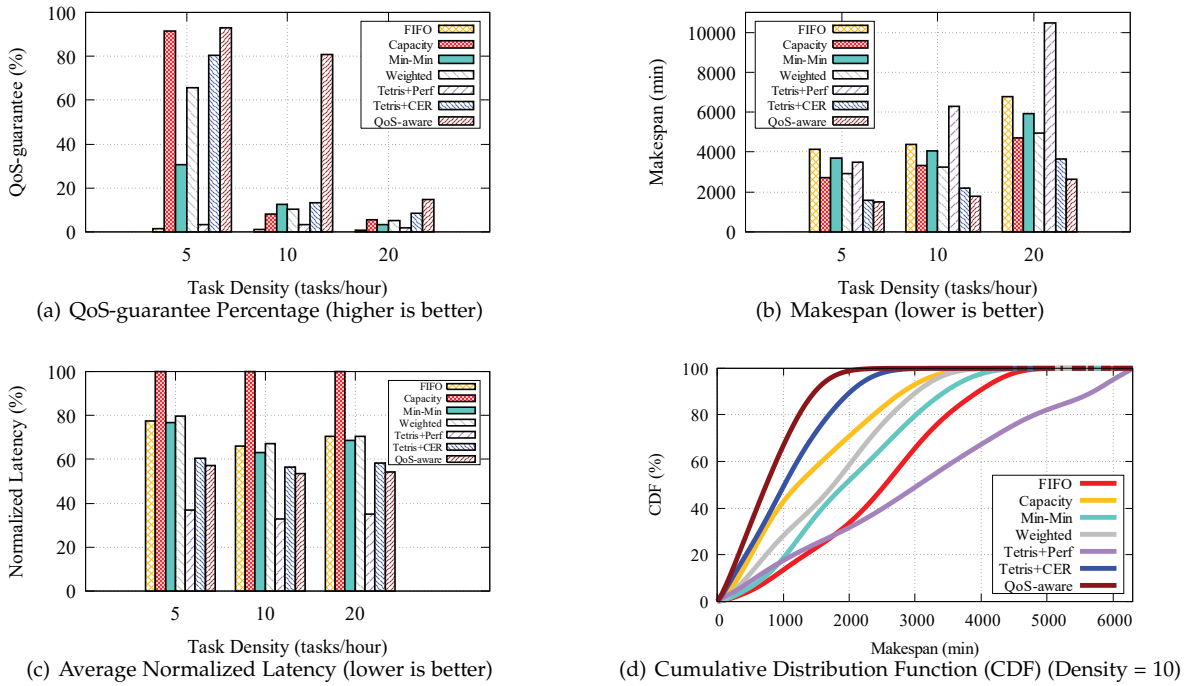
7.2 Evaluation of the Strategy on a GPU Cluster

7.2.1 Scheduling Performance and Comparison

We evaluate our proposed QoS-aware scheduling strategy using four baselines for comparison. As shown in Fig. 13(a) and Fig. 13(b), the results demonstrate that with an increasing task density, the QoS-guarantee percentage decreases and the makespan increases for all scheduling algorithms. Specifically, *Capacity Scheduler* and *Tetris+CER Scheduler* outperform other baselines. This may be because *Capacity Scheduler* and *Tetris+CER Scheduler* focus more on resource sharing than on single task performance. For *FIFO Scheduler* and *Tetris+Perf Scheduler*, the pursuit of a single task performance leads to additional waiting time for other tasks and low system utilization. *Min-Min Scheduler* and *Weighted Fair Scheduler* only focus on users' QoS, ignoring task performance. In this paper, our proposed QoS-aware strategy takes users' ECTs and the cost-effective model into consideration, achieving a trade-off between QoS guarantees for all tasks and system utilization. Moreover, the QoS-aware algorithm has a stronger tolerance and robustness for diverse task densities because the scheduling decision is made by the scheduler according to current users' QoS and system loads dynamically. As the results show, the QoS-aware algorithm achieves an improvement of up to 67.4% in QoS-guarantee percentage and a 28.2% reduction in makespan over the best baselines. However, when the task density is too high (> 20), users' QoS is difficult to guarantee due to limited computing resources, even for a QoS-aware algorithm. We present the average normalized latency of all tasks in Fig. 13(c). *Tetris+Perf Scheduler* can identify the most effective placement solutions. However, only maximizing the performance of a single task may lead to a long waiting time and a high makespan due to the resource contention of multiple tasks. However, *Tetris+CER Scheduler* and the QoS-aware strategies take user's QoS and resource sharing into consideration and achieve comparable average normalized latency under varying task densities. Instead, *FIFO Scheduler*, *Capacity Scheduler*, *Min-Min Scheduler* and *Weighted Fair Scheduler* treat DL tasks as black boxes and place tasks with simple heuristics, ignoring the relationship between resource provisions and task performance. At last, the Cumulative Distribution Function (CDF) in Fig. 13(d) demonstrates that the QoS-aware strategy and *Tetris+CER*

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558



(a) QoS-guarantee Percentage (higher is better)
(b) Makespan (lower is better)
(c) Average Normalized Latency (lower is better)
(d) Cumulative Distribution Function (CDF) (Density = 10)

Fig. 13. Performance Comparisons for Diverse Scheduling Strategies

Scheduler focus more on resource efficiency and achieve a higher task completion speed and system utilization than other scheduling methods.

7.2.2 Sensitivity Analysis

In this section, we examine how diverse workloads affect scheduling performance. Firstly, we focus on the impact of processing rate under varying task densities. As shown in Fig. 14(a) and Fig. 14(b), when the task density is sparse, most scheduling algorithms can achieve good performance. However, with the increasing task density, the users' QoS of all tasks are more difficult to guarantee due to the limitation of computing resources. Meanwhile, the makespans of all scheduling algorithms increase along with the growth of the task densities. Compared with other baselines, our proposed QoS-aware scheduling algorithm takes the users' QoS into consideration in the scheduling and has better tolerance for varying task densities, achieving improvements of up to 67.5% in the QoS-guarantee percentage and a 39.3% reduction in the makespan over the best baselines.

We further investigate the scheduling performance in various emergency scenarios. As shown in Fig. 14(c) and Fig. 14(d), the three numbers represent the proportions of different QoS priorities. The QoS-guarantee percentage decreases appreciably as the proportion of the high-priority tasks increases. Meanwhile, the QoS-aware scheduling algorithm attempts the most effective placement to guarantee users' QoS and decrease the makespan.

Finally, Fig. 14(e) and Fig. 14(f) show the sensitivities to the task arrival duration when the task density $= 15$. Although the makespan increases with the growth of the task arrival duration, the QoS-guarantee percentage only has an imperceptible decline. The results demonstrate that our proposed QoS-aware algorithm has a better adaptability than other baselines for long-term scheduling.

7.2.3 Accuracy of Prediction Model

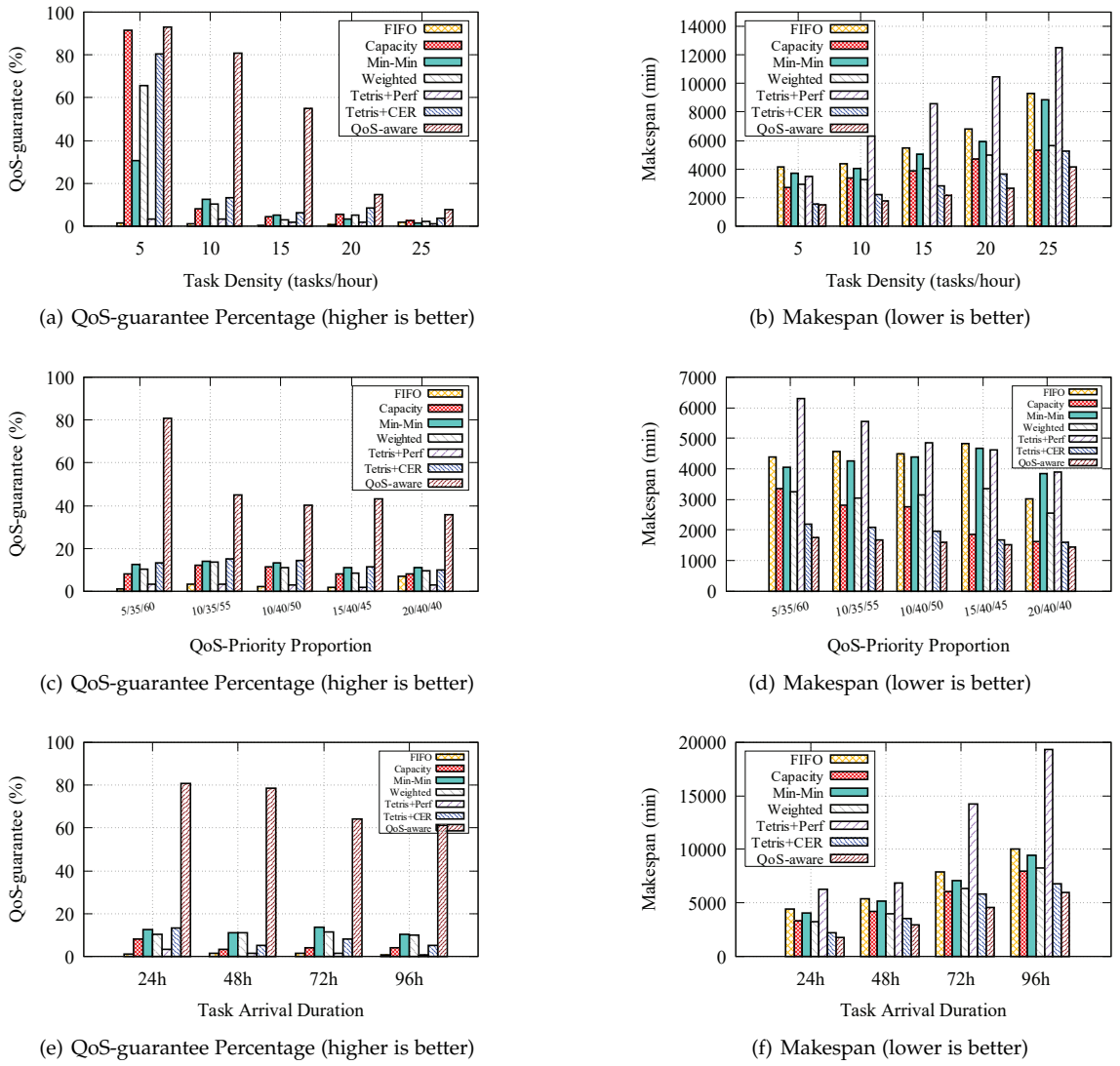
Our proposed QoS-aware scheduling algorithm relies on the prediction model for the response latency, as explained in Section 5.1. In this section, we evaluate the accuracy of the latency prediction model to validate its effectiveness. The measured latency of each task under diverse task densities are collected and compared with the predicted value, which is calculated based on Eq. 4 and Eq. 5. Fig. 15 shows the distribution of the prediction errors for all prediction cases. With the task density increasing, the average prediction error is higher due to the more intensive resource contentions and interferences among multiple tasks. However, the $95th$ percentile is $< 11\%$ even if the task density is high ($= 20$). Meanwhile, in our proposed QoS-aware scheduling algorithm, the prediction model is mainly adopted to analyze the variation trends and CERs of diverse task placements. The evaluation results demonstrate that the prediction models from the lightweight profiler are sufficiently accurate and the errors are insignificant in our scheduling design.

7.2.4 Scheduling Overhead

The scheduling overhead of diverse scheduling strategies is mainly derived from the monitoring system load and calculating task placements based on prediction models while the whole scheduler is running on the master node. As shown in Fig. 16, as the task density increases, the schedulers make more decisions concerning task placements, which leads to a higher scheduling overhead. Due to the complex scheduling design, our QoS-aware strategy has a higher cumulative scheduling overhead than other methods. Compared with the makespan ranging from dozens to hundreds of hours, the proportion of the scheduling overhead is less than 2%, even if the task density $= 20$. Therefore, we conclude

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558



(a) QoS-guarantee Percentage (higher is better)

(b) Makespan (lower is better)

(c) QoS-guarantee Percentage (higher is better)

(d) Makespan (lower is better)

(e) QoS-guarantee Percentage (higher is better)

(f) Makespan (lower is better)

Fig. 14. Sensitivity Analysis under Diverse Workloads. (a)(b) Sensitivity to Task Density. (c)(d) Sensitivity to QoS-Priority Proportion. (e)(f) Sensitivity to Task Arrival Duration.

that our proposed QoS-aware scheduling algorithm is sufficiently fast and the scheduling overhead can be ignored.

7.3 Large-scale Simulation

In order to evaluate the scalability of GENIE, we propose a trace-driven simulation on larger distributed clusters. Considering the increasing scale of clusters, the workload traces we adopted here are also generated by a Poisson distribution with a task density of 20 tasks/hour. The evaluation results are shown in Fig. 17. As the scales of the platform increase from 16 GPUs to 128 GPUs, all scheduling strategies have sufficient processing capabilities to meet users' QoS and a high task completion speed. The QoS-guaranteed percentage is improving and the makespans are decreasing for all scheduling strategies. However, a greater amount of computing resources can lead to more placement choices for each task. Compared with other baselines, the QoS-guided strategy can identify the most efficient placements to improve the QoS-guarantee percentage and maximize system utilization on any scale platform. Specifically, the QoS-guided strategy has achieved an improvement of up to 46.5% in QoS-guarantee percentage and a 20.8% reduction in makespan over the best baselines. According to the evaluation results, our proposed QoS-aware algorithm demonstrates a good scalability on diverse larger distributed clusters.

7.4 Discussion

7.4.1 *Training Accuracy*

For DL R&D platforms, several new model structures have been proposed for model and application explorations, called AutoML [38]. During training processes, convergence and accuracy are usually unpredictable in most cases. In this paper, we select data parallelism as a distributed approach on multiple devices [30, 32]. For a fixed global batch size, the more nodes and devices that are divided and placed, the smaller the local batch size is on each device. This distributed training method has been proven no impact on accuracy and convergence. In the future, our work can be

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

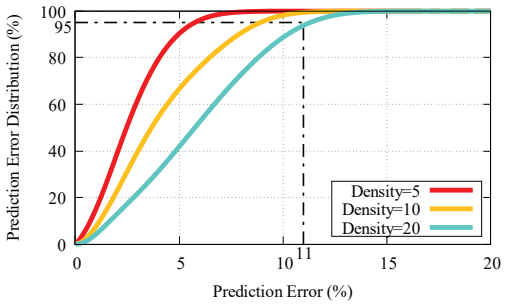


Fig. 15. Accuracy of Prediction Model. When the task density is high ($= 20$), the $95th$ percentile of prediction error is $< 11\%$.

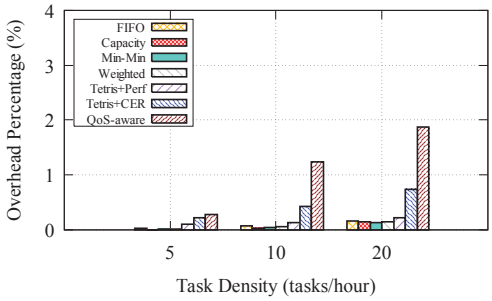
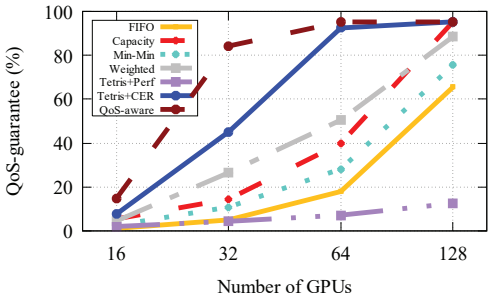
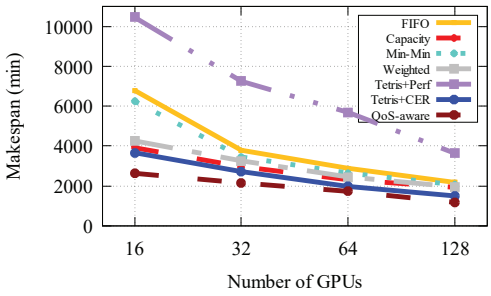


Fig. 16. Cumulative Scheduling Overhead on Master Node. Compared with the makespan ranging from dozens to hundreds of hours, the proportion of scheduling overhead is less than 2%.



(a) QoS-guarantee Percentage (higher is better)



(b) Makespan (lower is better)

Fig. 17. Performances of Large-Scale Simulation. The results demonstrate that the QoS-aware algorithm has better scalability on larger distributed clusters.

merged with an AutoML framework to further improve scheduling efficiency.

7.4.2 Multi-task Interference

Considering that GPUs are adopted as exclusive accelerators in this paper, the interference among multiple tasks derives from the contentions of communications bandwidths. As shown in Fig. 18, different communication paths can lead to different degrees of interference. Fig. 18(a) shows the intra-node interference for two single-GPU training tasks of *Inception-v3* on a single node. The background task is configured with a global batch size of 16. We observe that the degrees of interference decrease with the increasing global batch size for the foreground and the distance between the locations of two tasks. This is because the tasks with a large batch size are not sensitive to bandwidth contentions, since they have low communication frequency. Meanwhile, a spreading task placement can effectively reduce the interference of multiple tasks. A similar trend is observed for inter-node interference. Fig. 18(b) shows that two 4-GPU tasks are running on two 4-GPU nodes, where each task occupies two GPUs on each node. The interference includes a network connection among nodes. The background task is configured with a global batch size of 64. The impact of the interference increases as the global batch size of the foreground task decreases. However, in our proposed QoS-aware scheduling algorithm, for the tasks with a small batch size, our scheduling strategy chooses to place them more compactly, which leads to less communication and interference. Therefore, the multi-task interference is not a significant factor in our scheduling framework.

7.4.3 Asymmetric Cluster Scheduling

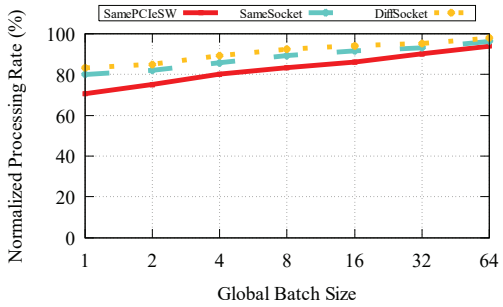
We assume that the GPU cluster is symmetric in this paper. However, scheduling on an asymmetric cluster is also an interesting issue for discussion. The differences of scheduling design between symmetric and asymmetric clusters includes: 1) Scheduling algorithm on asymmetric clusters should tap their respective advantages of different nodes to process diverse tasks. For example, the nodes with only 1 GPU can process the training task with small batch size while the nodes with multiple GPUs are suitable to process train DL models with big batch size. 2) Building prediction models and making placement decision should take more factors into consideration. For a symmetric cluster, the number of nodes and the number of GPUs in one node can represent the diverse placements. However, for an asymmetric cluster, the specific kind of node should be assigned to make the placement decision. A more fine-grained characterization and scheduling can be design for asymmetric clusters, which is a future research direction.

8 Related Work

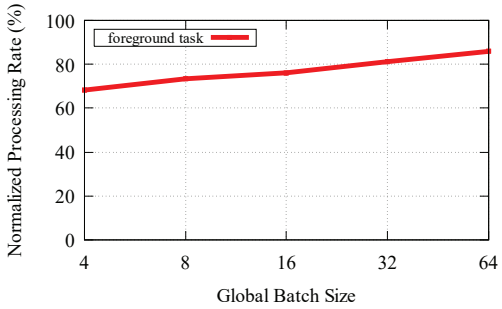
DL/ML Scheduling. DL and ML have been adopted as representative techniques in numerous AI challenges. As the number of DL/ML tasks increases in diverse platforms and scenarios, recent research has focused on the efficiency of a single task and the scheduling of multiple tasks [6, 7, 21–23, 25–27]. [21, 22] have attempted to explored commodity

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558



(a) Intra-Interference (two single-GPU tasks on a single node)



(b) Inter-Interference (two 4-GPU tasks on two nodes)

Fig. 18. Intra-node and inter-node interference for foreground tasks. In (a), the background task is configured with model *Inception-v3* and global batch size 16 on a single GPU. Moreover, *SamePCIeSW*, *SameSocket* and *DiffSocket* denotes two GPUs on the same PCIe Switch, two GPUs on the different PCIe Switch but on the same CPU socket and two GPUs on the different CPU socket, respectively. In (b), the background task is configured with global batch size 64 and four GPUs on two nodes.

GPU-based platforms to improve execution efficiency for multiple inference tasks. Compared to inference behavior, training tasks have more complex computation and communications. [23] designed a hierarchical model for efficient placement of a single DL training task based on reinforcement learning. [6, 7] presented novel workload placement strategies to schedule multiple training tasks on a single multi-GPU cluster. Recently, research [25–27] has proposed applying DL scheduling frameworks to fine-grained allocation of resources based on runtime performance, achieving low training latency and high cluster efficiency. However, our proposed scheduling framework, GENIE, can support diverse DL task types, including training and inference. Moreover, users' QoS requirements are considered in the design of our scheduling policy, which is closer to real DL R&D scenarios.

Cluster Scheduling. Most novel schedulers have been proposed for cloud computing or HPC systems [15, 16, 39, 40]. However, the jobs are treated as black boxes for these schedulers. Existing schedulers allocate resources based on historical information or simple heuristics methods. [15] focus on the dependency structures of tasks and adopt the heuristics method to pack and schedule them. [16] explores the similar patterns of diverse tasks based on historical information and scheduling related batch tasks to improve system utilization. Compared with the above mentioned research, the scheduling design in this paper is based on a comprehensive characterization for DL models. According to the domain-specific knowledge of DL tasks, our proposed scheduling policy can predict the relationship between resource provision and its impacts on task performance, achieving high system utilization.

Performance Estimation and Modeling. [22] made the first attempt to explore the performance of DL tasks on a commodity GPU-based platform and has benefits for future WSC design. [26] proposed performance models to predict the model convergence and runtime by online monitoring. In this paper, considering the unpredictability of convergence and accuracy for online training scenarios, we prefer modeling the processing rate and response latency of DL tasks. Based on a lightweight offline profiler, the prediction model can achieve high prediction accuracy and provide support for our QoS-aware scheduling algorithm.

QoS Scheduling. [41] presents a combination of profiling and job structure knowledge for scheduling to meet users' QoS automatically. [42] proposes a QoS scheduling solution by dynamic provision based on performance models derived from historical and dependency information. In our work, considering the emergencies of varying degrees, we define three classes of users' QoS. A high user's QoS means high priority for resource allocation and execution. During the whole scheduling process, the scheduling order is dynamically adjusted according to the users' QoS in order to achieve a high QoS guarantee.

9 Conclusion

This paper presents GENIE, a QoS-aware scheduling framework for a DL R&D platform in most EIUs. GENIE provides lightweight offline profiling and online dynamic scheduling on GPU clusters. Using the lightweight offline profiler, GENIE can provide a prediction model according to the domain-specific information of DL tasks derived from a comprehensive characterization. Based on the prediction models, GENIE dynamically identifies the best placements for DL tasks and schedules them on the GPU cluster. Numerous experiments on real clusters and simulations demonstrate that GENIE achieves a higher QoS guarantee and system utilization than other baselines. In the future, we plan to merge GENIE into cluster manager frameworks like Kubernetes, Yarn, or Mesos.

Acknowledgments

The authors gratefully acknowledge supports from National Key Research and Development program under No.2016YFB1000400, 2018YFB0204300; National Nature Science Foundation of China under NSFC No. 61872377, 61802417 and 61802420; NUDT Science Foundation under No. ZK18-03-40.

References

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2012.

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

[2] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro and et al., "Deep speech 2: end-to-end speech recognition in English and mandarin," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

[3] Amazon Machine Learning - Predictive Analytics with AWS. https://aws.amazon.com/machine-learning/, 2017.

[4] Google Cloud Prediction API Documentation. https://cloud.google.com/prediction/docs/, 2017.

[5] Machine Learning - Predictive Analytics with Microsoft Azure. https://azure.microsoft.com/en-us/services/machine-learning/, 2017.

[6] M. Amaral, J. Polo, D. Carrera, S. R. Seelam and M. Steinder, "Topology-aware GPU scheduling for learning workloads in cloud environments," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2017.

[7] Z. Chen, L. Luo, W. Quan, Y. Shi, J. Yu, M. Wen and C. Zhang, "Multiple CNN-based Tasks Scheduling across Shared GPU Platform in Research and Development Scenarios," in *Proceedings of IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2018.

[8] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan and et al., "Machine Learning at Facebook: Understanding Inference at the Edge," in *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[9] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov and et al., "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective," in *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[10] NVIDIA DGX Station. https://www.nvidia.com/en-us/data-center/dgx-1/.

[11] Meet Michelangelo: Uber's Machine Learning Platform. https://eng.uber.com/michelangelo/.

[12] D. Baylor, E. Breck, H. Cheng, N. Fiedel, C. Foo, Z. Haque and et al., "TFX: A TensorFlow-Based Production-Scale Machine Learning Platform," in *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

[13] V. Sridhar, S. Subramanian, D. Arteaga, S. Sundararaman, D. Roselli and N. Talagala, "Model Governance: Reducing the Anarchy of Production ML," in *Proceedings of USENIX Annual Technical Conference (ATC)*, 2018.

[14] Tianhe-2 Supercomputer. http://nscc-gz.cn/.

[15] R. Grandl, M. Chowdhury, A. Akella and G. Ananthanarayanan, "Altruistic Scheduling in Multi-Resource Clusters," in *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[16] Y. Zhang, G. Prekas, G. M. Fumarola, M. Fontoura, I. Goiri and R. Bianchini, "History-Based Harvesting of Spare Cycles and Storage in Large-scale Datacenters," in *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[17] Graphite Cluster of Cornell University. https://graphite.coecis.cornell.edu/ganglia/.

[18] MSEAS Cluster of MIT Cluster. http://mseas.mit.edu/ganglia/.

[19] Tiger Cluster of UC Berkeley. http://tiger.cchem.berkeley.edu/ganglia/.

[20] M. Jeon, S. Venkataraman, J., A. Phanishayee, W. Xiao and F. Yang, "Multi-tenant GPU clusters for deep learning workloads: Analysis and implications," in *Microsoft Report*, 2018.

[21] F. Yan, Y. He, O. Ruwase and E. Smirni, "SERF: efficient scheduling for fast deep neural network serving via judicious parallelism," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2016.

[22] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. N. Mudge, R. G. Dreslinski, J. Mars and L. Tang, "DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2015.

[23] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V Le and J. Dean, "A Hierarchical Model for Device Placement," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[24] Y. You, Z. Zhang, C. Hsieh, J. Demmel and K. Keutzer, "ImageNet Training in Minutes," in *Proceedings of International Conference on Parallel Processing (ICPP)*, 2018.

[25] W. Xiao, R. Bhardwaj , R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang and L. Zhou, "Gandiva: Introspective Cluster Scheduling for Deep Learning," in *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[26] Y. Peng, Y. Bao, Y. Chen, C. Wu and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *Proceedings of the EuroSys Conference (EuroSys)*, 2018.

[27] Z. Chen, L. Luo, H. Yang, J. Yu, M. Wen and C. Zhang, "GENIE: QoS-guided Dynamic Scheduling for CNN-based Tasks on SME Clusters," in *Proceedings of Design Automation and Test in Europe (DATE)*, 2019.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[29] I. Sutskever, O. Vinyals and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2014.

[30] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," in *CoRR abs/1404.5997*, 2014.

[31] Tensorflow Using GPUs. https://www.tensorflow.org/guide/usings_gpu.

[32] Caffe Multi-GPU-Usage. https://github.com/BVLC/caffe/blob/master/docs/multigpu.md.

[33] M. Song, Y. Hu, H. Chen and T. Li, "Towards Pervasive and User Satisfactory CNN across GPU Microarchitectures," in *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[34] 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

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2019.2931558

Warehouse-Scale Computers," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[35] T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther and et al., "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," in *Journal of Parallel and Distributed Computing (JPDC)*, vol. 61, no. 6, pp. 810-837, 2001.

[36] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker and I. Stoica, "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[37] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao and A. Akella, "Multi-resource packing for cluster schedulers," in *Proceedings of the International Conference on the applicaitonss, technologies, architectures, and protocols for computer communication (SIGCOMM)*, 2014.

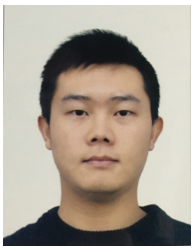
[38] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *CoRR abs/1611.01578*, 2016.

[39] R. Pathan, P. Voudouris and Per Stenström, "Scheduling Parallel Real-Time Recurrent Tasks on Multicore Platforms," in *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 29, no. 4, pp. 915-928, 2018.

[40] J. Zhu, X. Li, R. Ruiz and X. Xu, "Scheduling Stochastic Multi-Stage Jobs to Elastic Hybrid Cloud Resources," in *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 29, no. 6, pp. 1401-1415, 2018.

[41] Z. Zhang, L. Cherkasova, A. Verma and B. T. Loo, "Automated profiling and resource management of pig programs for meeting service level objectives," in *Proceedings of International Conference on Autonomic Computing (ICAC)*, 2012.

[42] A. D. Ferguson, P. Bodík, S. Kandula, E. Boutin and R. Fonseca, "Jockey: guaranteed job latency in data parallel clusters," in *Proceedings of the EuroSys Conference (EuroSys)*, 2012.



Zhaoyun Chen received the B.S. and M.S. degrees in 2013 and 2015 from the National University of Defense Technology, Changsha, China, where he is currently working toward the Ph.D. degree with the College of Computer. His research interests focus on distributed and parallel computing, resource management and workload scheduling.

Wei Quan is an asistant professor in the College of Computer at National University of Defense Technology in China. His research interests include computer architecture, network architecture, design and optimization of multi-/many-core systems (in particular Multiprocessor System-on-Chip systems), design space exploration and resource scheduling.

Mei Wen is currently a professor at the Computer College at the National University of Defense Technology, China. She received her BS, MS, and PhD in Computer Science and Technology from the National University of Defense Technology in 1995, 1999 and 2006, respectively. Her research interests include computer architecture, parallel programming, and scientific computing.

Jianbin Fang is currently an Assistant Professor in computer science at National University of Defense Technology, Changsha, China. He obtained his Ph.D. in 2014 from the Parallel and Distributed System Group at Delft University of Technology, the Netherlands. He is a member of ACM and CCF. His general research area is parallel computing and in particular includes the topics of parallel programming and performance optimizations on multi-/many-cores, source-to-source compilers, performance modeling and prediction with statistical learning, and scalable parallel algorithms.

Jie Yu received his Ph.D. degree and now is an associate researcher in NUDT (National University of Defense Technology), China. He focuses on System Software, including security and performance in AI (mostly on Image Recognition and NLP) and OS (mostly on Linux and Android). He has published more than 50 papers on international conferences and journals, including SIGCOMM, ICPP, SecureComm, ISPA etc.

Chunyuan Zhang is a professor at the Computer College at the National University of Defense Technology, China. He received his BS, MS, and PhD in Computer Science and Technology from the National University of Defense Technology in 1985, 1990, and 1996, respectively. He is the director of a series of research projects including National Natural Science Foundation projects of China. His research interests include computer architecture, parallel programming, embedded systems, and scientific computing.

Lei Luo received his B.S., M.S., and Ph.D. degrees from College of Computer, National University of Defense Technology, China, in 2006, 2008, and 2013, respectively. He joined College of Computer, National University of Defense Technology, as a Lecturer, in 2013. His research interests include computer vision, system software, and machine learning.

Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.