Grouper: Accelerating Hyperparameter Searching in Deep Learning Clusters With Network Scheduling

Pan Zhou, Hongfang Yu, and Gang Sun, *Member, IEEE*

***Abstract*—Training a high-accuracy model requires trying hundreds of configurations of hyperparameters to search for the optimal configuration. It is common to launch a group of training jobs (named cojob) with different configurations at the same time and stop the jobs performing worst every stage (i.e., a certain number of iterations). Thus deep learning requires minimizing stage completion time (SCT) to accelerate the searching. To quickly complete the stages, each job in the cojob typically uses multiple GPUs to perform distributed training. The GPUs exchange data per iteration to synchronize their models through the network. However, data transfers of DL jobs compete for network bandwidth since the GPU cluster hosts a number of cojobs from various users, resulting in network congestion and consequently a large SCT for cojobs. Existing flow schedulers aimed at reducing flow/coflow/job completion time mismatch the requirement of hyperparameter searching. In this paper, we implement a system Grouper to minimize average SCT for cojobs. Grouper adopts a well-designed algorithm to permute stages of cojobs and schedules flows from different stages in the order of the permutation. The extensive testbed experiments and simulations show that Grouper outperforms advanced network designs Baraat, Sincrona, and per-flow fair share.**

***Index Terms*—Deep learning, hyperparameter search, flow scheduling, stage completion time.**

I. Introduction

RECENTLY most leading companies, such as Google and Microsoft, have applied deep learning (DL) models in various products [1], [2]. They have built large GPU clusters to facilitate model training. However, training a high-accuracy DL model is not a one-time effort. It typically requires training the same model multiple times with various configurations of hyperparameters (e.g., the learning rate and the model structure) to search for one configuration that can optimize the model accuracy [3]. Due to the large searching space, the model will be retrained hundreds to thousands of times [4], [5]. To speed up the searching process, it is common to launch a group of training jobs (a *cojob*) with different configurations at a time and, at each stage (e.g., every 1000 mini-batch iterations), kill a subset of the jobs with the lowest accuracy [6], [7]. Finally, only one job with the highest accuracy will be allowed to run to completion. Such a stage-based searching process requires minimizing average stage completion time (SCT) of cojobs to stop jobs with unpromising configurations earlier and then free the occupied resources to try new configurations faster.

To quickly complete the stages, every training job in a cojob typically utilizes multiple GPUs to perform distributed training where the GPUs exchange information (e.g., models and gradients) through the network per iteration [8]. Consequently, each job will produce many data flows in the network over time and hence the network bandwidth is vital for DL jobs to quickly complete the stages. However, there are a number of cojobs running in the GPU cluster. The data flows of DL jobs in various cojobs compete for the limited bandwidth in the network [2]. Due to the large sizes of models (e.g., hundreds of megabytes), data sizes of the flows are also large and the network competition of the flows can easily cause network congestion [7], which significantly increases average SCT for cojobs. Thus an efficient network scheduling is desired to reduce the average SCT.

Unfortunately, state-of-the-art network scheduling designs mismatch the objective of minimizing average SCT for cojobs. For example, flow-level schedulers, such as Eiffel [9], PIAS [10] and s-PERC [11], focus on minimizing average flow completion time by considering flow-level information (e.g., flow size). And coflow-level schedulers, such as PRO [12] and Sincrona [13], focus on minimizing average coflow completion time by taking coflow-level information (i.e., flow number and flow size in a coflow) into consideration. While, a stage of a cojob involves multiple DL jobs and each of these jobs will finish a number of iterations at this stage. Since the data flows of one iteration can be abstracted as a coflow, each of the involved jobs will generate a series of coflows at this stage. Consequently, the stage size is not only dependent on the size of flows/coflows but also the number of coflows; the flow-/coflow-level schedulers are agnostic to such stage-level information and may greedily schedule jobs with shorter flows or coflows. As a result, shorter stages may be blocked by longer stages, which significantly increases SCT. But naively

Manuscript received January 23, 2020; revised April 8, 2020; accepted April 17, 2020. Date of publication April 21, 2020; date of current version September 9, 2020. This work was supported by the National Key Research and Development Program of China under Grant 2019YFB1802800, PCL Future Greater-Bay Area Network Facilities for Large-Scale Experiments and Applications under Grant PCL2018KP001, Fundamental Research Funds for the Central Universities under Grant ZYGX2016J217, and the Fundamental Research Funds for the Central Universities under Grant 2682019CX61. The associate editor coordinating the review of this article and approving it for publication was N. Zincir-Heywood. *(Corresponding authors: Hongfang Yu; Gang Sun.)*

Pan Zhou and Gang Sun are with the Key Laboratory of Optical Fiber Sensing and Communications, Ministry of Education, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: gangsun@uestc.edu.cn).

Hongfang Yu is with the Key Laboratory of Optical Fiber Sensing and Communications, Ministry of Education, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with Peng Cheng Laboratory, Shenzhen 518066, China (e-mail: yuhf@uestc.edu.cn).

Digital Object Identifier 10.1109/TNSM.2020.2989187

applying a shortest-first heuristic, the predominant way to solve most scheduling problems, is not effective to minimize average SCT for cojobs, because the stages in a cojob have precedence constraints and network flow scheduling involves coupled resources: a network has a number of ingress and egress ports and each flow's progress is dependent on its rates at both ingress and egress port.

In this paper, we research the problem of network scheduling to minimize average SCT for cojobs in GPU clusters. We formulate the problem as an optimization model, which is nonlinear programming with infinite variables and is proved to be strongly NP-Hard. It cannot be directly resolved. Accordingly, we propose a cojob-aware permutation-and-schedule method to approximately solve this problem with an approximation ratio of 2x. In this method, a primal-dual algorithm is well designed to permute stages of cojobs, and flows belonging to different stages are scheduled in the order of their stage permutation. Moreover, we also implement the method in a real system Grouper. Grouper utilizes the priority queues of the existing network to schedule flows rather than complex rate allocation. Specifically, for any given flow, Grouper offload it into the corresponding priority queue according to its stage order in the permutation. Obviously, this implementation is readily-deployment and causes little overhead.

We evaluate Grouper through small-scale testbed implementation and large-scale simulations. Our extensive experiments and simulations show that Grouper can reduce average SCT by up to 22.5%, 24.2%, and 31.0%, compared to advanced network designs Baraat, Sincrona, and per-flow fair share (FS), respectively. The corresponding saving of job completion time is 16.3%, 26.5%, and 28.0%, respectively.

Generally, the major contributions are as follows:

- We propose the problem of scheduling flows to minimize average SCT of cojobs for accelerating the hyperparameter search process.
- We formulate the problem as an optimization model and analyze its hardness.
- We design a fast and efficient permutation-and-schedule method to approximately solve the problem with 2x approximation ratio.
- We implement our method in a real system Grouper. Our extensive testbed experiments and simulations show that Grouper outperforms state-of-the-art network designs.

The rest of the paper is organized as follows. Section II overviews the background and the motivation of network scheduling to reduce SCT for cojobs. Section III provides the design and mathematical analysis of our scheduling method. Then Section IV implements the proposed method into a real system Grouper. Section V evaluates Grouper through extensive testbed experiments and digital simulations. Sections VI and VII provide the most related work and the conclusion of the paper, respectively.

II. Background and Motivatoin

A. Distributed Model Training

Due to the ever-growing large scale of datasets, it is common to use parameter server architecture to realize data parallelism training across multiple workers (machines or GPUs) [14]–[16]. As shown in Fig. 1, the entire dataset is split among multiple workers and each worker trains a local copy of the global model with its allocated data partition; the global model is also partitioned to be maintained by multiple parameter servers (PS). At each iteration, each worker calculates parameter updates (i.e., gradients) of the local model over a batch of training data (i.e., a mini-batch), then pushes the updates to PSs maintaining corresponding parameters. After receiving updates from all workers, PSs update the parameters of the global model with an optimization algorithm, e.g., the stochastic gradient descent (SGD) or its variants. Then workers pull the updated parameters from PSs to update their local copy and start the calculation of the next iteration. Through such a process to exchange data with PSs, distributed workers can synchronize their model parameters to ensure the correctness of training. Typically, to achieve high accuracy, thousands to millions of such iterations are performed on the training dataset, resulting in multiple epochs/passes over the entire dataset [2], [7].

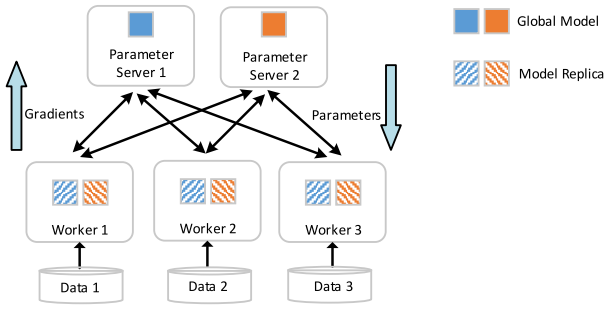


Fig. 1. Parameter server architecture to realize data parallelism training. This figure shows two parameter servers and three workers.

B. Hyperparameter Searching

The final performance (i.e., accuracy or loss) achieved by a trained DNN model highly depends on many adjustable hyperparameters, such as the learning rate of SGD, the batch sizes, number and size of hidden layers in a DNN. Unfortunately, these hyperparameters cannot be easily set by standard optimization methods since the effect of interaction between hyperparameters on the model quality is unknown so far. Thus all the work of hyperparameter tuning depends on empirical trials to search desired configurations of hyperparameters that optimize the model performance. However, the search space of hyperparameters is very huge and the training of each individual configuration takes a long time, which makes the hyper-parameter tuning work ultra-time-consuming and resource-costing. For example, prior research shows that it can take up to ten days to train an ImageNet22k image classification model to convergence even using 62 machines [3]. Therefore, the research of efficient hyperparameter searching to optimize model quality, reduce training time, and improve the efficiency of cluster resources has attracted much attention. The research effort is mainly focused on the following two aspects.

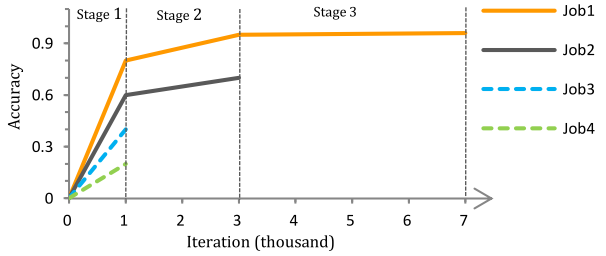


Fig. 2. An illustration of Successful-Halving algorithm. The cojob has 4 DL jobs and is divided into 3 stages.

The first aspect is to research the techniques for generating candidate configurations from the large scale search space. Traditional brute-force methods like random search [17] and grid search [18] are simple and easy to use. Recently adaptive techniques, such as Bayesian optimization methods [4], are considered to be more efficient than random and grid search.

The second aspect is to research the techniques for accelerating the evaluation of the candidate configurations. Bandit-based algorithm and its variants, such as Successful-Halving [19], HyperDrive [3] and HyperBand [5], have been widely adopted because of their powerful performance and extreme simplicity [6]. All these algorithms typically launch a group of jobs (a cojob) with different configurations at a time. The cojob is divided into multiple stages each of which contains a certain number of iterations. At the end of each stage, a portion of jobs performing poorly will be stopped and the remaining jobs will continue a given number of iterations in the next stage. For example, Successful-Halving algorithm stops the worst half of the remaining jobs, double the iteration number of next stage for each remaining job and successively repeats until only a single job is left. This process is illustrated in Fig. 2 where a cojob has 4 jobs. Its first stage contains 1000 iterations, after all jobs finish 1000 iterations, the two jobs performing worst are stopped and the other two jobs continue in the next stage. Through such a stage-based process, the bad configuration jobs can be stopped early rather than training them converged, and then a new cojob can be launched early. Therefore, minimizing stage completion time is vital for reducing the total searching time as well as resource cost.

C. Motivation

Due to the ever-growing number of AI-based services, most leading companies have built GPU clusters to train various DNN models [1], [2], [20]. For resource efficiency, the clusters are shared by many engineers from different product teams. Thus various cojobs are submitted to the cluster over time. As mentioned above, the model synchronizations per-iteration among PSs and workers produce data flows in the network fabric. As the model sizes are typically ultra-large (i.e., hundreds of megabytes), the data flows belonging to different cojobs compete for network bandwidth, leading to heavy network congestion and thus resulting in long synchronization time per iteration as well as long stage completion times (SCT). Moreover, the computing power of hardware accelerators is growing much faster than network speed. The workers can produce data flows (or parameter updates) faster than can be naively synchronized over the network, which makes the network fabric become a well-known performance bottleneck for distributed deep learning [21].

For example, the research of Poseidon [22] shows that when training AlexNet model (about 61.5M parameters) on Titan X GPUs, each GPU worker requires throughput larger than 26Gbps to make sure the next iteration of computation not being blocked. Today's GPU server usually contains 8 or more GPUs. Thus a GPU server in the shared cluster typically supports multiple workers or parameter servers belonging to different training jobs. The total throughput required by the workers at the same server can reach to $26 \times 8 = 208$ Gbps if there are 8 GPU workers running at the server. The required throughput obviously exceeds the bandwidth that advanced Ethernet (i.e.,10Gbps and 100Gbps Ethernet) provides. And the research of Microsoft [2], [20] shows that the network competition of GPU workers at the same server can significantly impact the training speed. Therefore, an efficient flow scheduler is desired to manage bandwidth competitions and reduce stage completion times for cojobs.

Advanced flow schedulers that aimed at improving flow/coflow completion times for distributed cloud applications [9], [11], [12] are ineffective in reducing average SCT for DL cojobs. The reason is that stage sizes are not proportional to flow-/coflow-sizes, these flow schedulers may greedily schedule jobs with shorter flows/coflows, which can let the longer stages block the shorter stages and consequently result in a long SCT. Advanced flow schedulers [23], [24] that aimed at minimizing job completion time are also ineffective in reducing SCT since they are agnostic to the precedence constraints of cojob-stages and the job interdependencies in a cojob-stage. Consider the example illustrated in Fig. 3(a). There are two DL cojobs start at the same time and share a single bottleneck link that can process one unit data at a time step. With per-flow fair share the standard bandwidth allocation schemes of today's datacenters [25], the result is illustrated in Fig. 3(b). In this case, all the flows of unfinished jobs obtain the same bandwidth, and cojob stages <A-1, A-2, B-1, B-2> are completed at time steps <4, 7, 9, 12>. Their average SCT is 8, which is 1.23x larger than the optimal as shown in Fig. 3(d). Similarly, SPTF [26], near optimal method to minimize average job completion time (JCT), can achieve average SCT 7.5 as shown in Fig. 3(c), which is 1.15x larger than the optimal.

The above observations motivate us to design a cojob-aware scheduler to minimize average SCT for distributed deep learning in GPU clusters.

III. Cojob-Aware Scheduling Algorithm

We first present the original formulation of the cojob-ware network scheduling problem and prove its NP-hardness in Section III-A. Then we relax it to linear programming in Section III-B. Based on the relaxation, we design a primal-dual algorithm to determine the order in which stages from different cojobs are scheduled across the network in Section III-C.

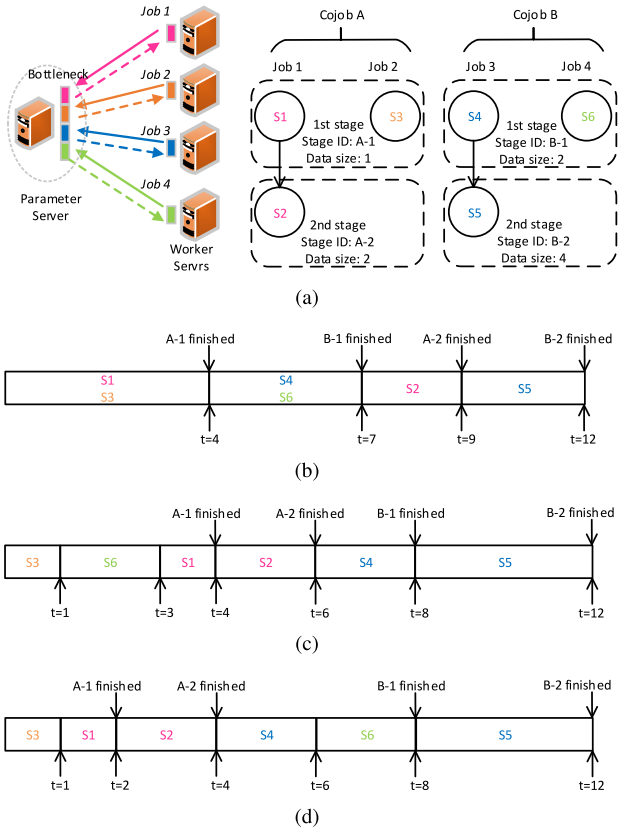


Fig. 3. An illustration of motivation. (a) Motivation settings: 2 cojobs, each of which contains 2 jobs, compete for a single bottleneck link; Job 1 has 1 and 2 unit of data in the first and second stage, respectively; Job 2 only has 1 unit of data in the first stage; Job 3 has 2 and 4 unit of data in the first and second stage, respectively; Job 4 only has 2 unit of data in the first stage; (b) Per-flow fair share (FS), average SCT is $\frac{4+7+9+12}{4} = 8$; (c) Shortest processing time first (SPTF), average SCT is $\frac{4+6+8+12}{4} = 7.5$; (d) The optimal, average SCT is $\frac{2+4+8+12}{4} = 6.5$.

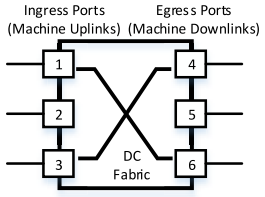


Fig. 4. Abstraction of cluster fabric.

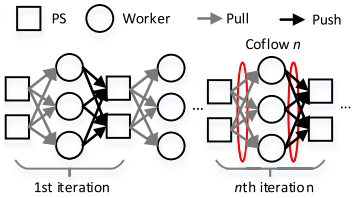


Fig. 5. Coflow abstraction for parameter server architecture.

TABLE I
NOTATIONS OF CONSTANTS

Symbol	Description
$\mathbb{G} = \{1, 2, ..., G\}$	The set of cojobs
K_g	The stage number of cojob g
N_g	The job number in cojob g
$I_{g,max}$	Maximum iteration number in cojob g
$I_{g,k}$	Total iteration number of first k stages for a job in cojob g. Note: $I_{g,K_g} = I_{g,max}$.
$\mathbb{U}^*_{g,k} = \{U^*_{g,k} \| k \in [K_g]\}$	The set of the first iteration for each stage in cojob g. Note: $U^*_{g,k} = I_{g,k-1} + 1$.
$\mathbb{P} = \{1, 2, ..., 2P\}$	The port set of fabric
$\mathbb{I} = \{1, 2, ..., P\}$	The ingress port set of fabric
$\mathbb{E} = \{P+1, ..., 2M\}$	The egress port set of fabric
T^*_g	The start time of cojob g
T	The completion time of cojobs in $\mathbb{G}$
$f^{g,j,i}_{s,d}$	The flow $(s \to d)$ of ith coflow of job j belonging to cojob g

Finally we provide our high-level idea of order scheduling scheme in Section III-D.

A. Problem Formulation

We first consider the offline scheduling of several cojobs on a non-blocking cluster fabric with P physical machines. As each machine contains one uplink and one downlink, the cluster fabric can be abstracted as a big switch containing P ingress ports and P egress ports. We assume that the ingress ports are indexed from 1 to P and the egress ports are indexed from $P+1$ to $2P$. All ports are assumed to have the same bandwidth capacity that is normalized to one. Such an abstraction is widely adopted in academic researches [13], [24] because this abstraction is based on the widely application of state-of-the-art Clos topologies, such as Leaf-Spine [27] and Fat-Tree [28]. The abstraction of a 3-machine cluster fabric is illustrated in Fig. 4.

With the abstraction of fabric, each distributed training job in a cojob can be abstracted as a set of coflows with precedence constraints. The abstraction of coflows for a job is illustrated in Fig. 5. For each iteration, workers pull the latest model parameters from PSs, then start computing model updates and push the updates to PSs. Thus the pull and push operations per iteration are abstracted as a coflow, as shown in Fig. 5. Without loss of generality, we assume that each coflow involves $P \times P$ parallel flows among the P ingress ports and P egress ports. For job j belonging to cojob g, the coflow produced by its nth iteration is a set of flows $D^{g,j,n} = \{d^{g,j}_{s,d} | 1 \le s \le P, P+1 \le d \le 2P\}$, where $d^{g,j}_{s,d}$ is the size of flow transferred from ingress port s to egress port d. Note that $d^{g,j}_{s,d}$ is equal to the size of model partition maintained at corresponding PS connecting to port s or d, and it does not vary with the iterations. In addition, if job j has no data transferred from port s to d, $d^{g,j}_{s,d}$ is set to 0.

In the mathematical analysis, we ignore the time interval between two consecutive data transfers (i.e., the pull and push operations) within the same job since the time interval is relatively small and the time of network communication dominates the training time. Then the network scheduling problem in deep learning clusters can be formulated as follows, and the notations of constants and variables we used are presented in Tables I and II.

Given a set of cojobs $\mathbb{G} = \{1, 2, 3, \ldots, G\}$, let $S_{g,k}$ be the completion time of kth stage of cojob $g \in \mathbb{G}$. Minimizing average completion times of cojobs' stages is equivalent to

TABLE II
NOTATIONS OF VARIABLES

Symbol	Description
$S_{g,k}$	Completion time of kth stage of cojob g
$E_{g,j,k}$	Completion time of kth stage of job j belonging to cojob g
$C_{g,j,i}$	Completion time of ith coflow of job j belonging to cojob g
$F_{s,d}^{g,j,i}$	The completion time of flow $f_{s,d}^{g,j,i}$
$t_{g,j,i}^{*}$	The start time of ith coflow of job j belonging to cojob g
$r_{s,d}^{g,j,i}(t)$	The transmission rate at time t of flow $f_{s,d}^{g,j,i}$
$d_{s,d}^{g,j}$	Total bytes of flow $f_{s,d}^{g,j,i}$, for all $i \in [I_{g,max}]$

minimizing the sum of their completion times, as shown in Eq. (1). For a given cojob g, let K_g denote its divided stage number, the stages are indexed by 1,2,..., K_g, and let $I_{g,k}$, $k \in \{1, 2, \ldots, K_g\}$ *or* $[K_g]$, denote the total iteration (or coflow) number of first *k* stages for each job in cojob *g*. The completion time of *k*th stage of a given cojob can be defined as the time when all jobs in the cojob have finished their *k*th stage, as shown in Eq. (2). And the *k*th stage of a given job is completed when all iterations (or coflows) in the first *k* stages are completed, as shown in Eq. (3). Without loss of generality, if the number of a given job's stages is less than *k*, we assume that its *k*th stage is completed at the time when it is stopped. For a given iteration or coflow, it is completed when all the flows at this iteration are completed, as shown in Eq. (4). The completion time of a given flow is defined by Eq. (5). In a given job, the next coflow cannot start until the current coflow is completed, as shown in Eq. (6) and (7). Note that we assume that all jobs in the same cojob will start next stage at the same time and all jobs arrival at time 0, as shown in Eq. (8) and (9), respectively. Moreover, the total data rate cannot exceed the capacity of each egress and ingress port at any time, as shown in Eq. (10) and (11). And Eq. (12) guarantees that the data rate for each flow must be non-negative. Such a coflow-level formulation presents the detailed inter-dependencies of coflows within a cojob-stage and inter-dependencies of coflows from different cojob-stages.

$$(\mathcal{O}) \quad \min \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} S_{g,k} \tag{1}$$

Subject to:

$$S_{g,k} = \max_{j\in[N_g]} E_{g,j,k},\ \forall g \in \mathbb{G}, k \in \left[K_g\right] \tag{2}$$

$$E_{g,j,k} = \max_{i\in\left[I_{g,k}\right]} C_{g,j,i},\ \forall g \in \mathbb{G}, j \in \left[N_g\right], k \in \left[K_g\right] \tag{3}$$

$$C_{g,j,i} = \max_{s,d\in\mathbb{P}} F_{s,d}^{g,j,i},\ \forall g \in \mathbb{G}, j \in \left[N_g\right], i \in \left[I_{g,max}\right] \tag{4}$$

$$\int_{t_{g,j,i}^{*}}^{F_{s,d}^{g,j,i}} r_{s,d}^{g,j,i}(t)dt = d_{s,d}^{g,j},$$
$$\forall s \in \mathbb{I}, d \in \mathbb{E}, g \in \mathbb{G}, j \in \left[N_g\right], i \in \left[I_{g,max}\right], t \in [T] \tag{5}$$

$$\int_{0}^{t_{g,j,i}^{*}} r_{s,d}^{g,j,i}(t)dt = 0,$$
$$\forall s \in \mathbb{I}, d \in \mathbb{E}, g \in \mathbb{G}, j \in \left[N_g\right], i \in \left[I_{g,max}\right], t \in [T] \tag{6}$$

$$t_{g,j,i}^{*} \geq C_{g,j,i-1},\ \forall g \in \mathbb{G}, j \in \left[N_g\right], i \in \left[I_{g,max}\right] \tag{7}$$

$$t_{g,j,i}^{*} = t_{g,j^{'},i}^{*},\ \forall g \in \mathbb{G}, j, j^{'} \in \left[N_g\right], i \in \mathbb{U}_{g,k}^{*} \tag{8}$$

$$C_{g,j,0}^{*} = T_{g}^{*} = 0,\ \forall g \in \mathbb{G}, j \in \left[N_g\right] \tag{9}$$

$$\sum_{g\in\mathbb{G}} \sum_{j\in[N_g]} \sum_{i\in[I_{g,max}]} \sum_{s\in\mathbb{I}} r_{s,d}^{g,j,i}(t) \leq 1,\ \forall d \in \mathbb{E}, t \in [T] \tag{10}$$

$$\sum_{g\in\mathbb{G}} \sum_{j\in[N_g]} \sum_{i\in[I_{g,max}]} \sum_{d\in\mathbb{E}} r_{s,d}^{g,j,i}(t) \leq 1,\ \forall s \in \mathbb{I}, t \in [T] \tag{11}$$

$$r_{s,d}^{g,j,i}(t) \geq 0, \forall s \in \mathbb{I}, d \in \mathbb{E}, g \in \mathbb{G}, i \in \left[I_{g,max}\right], t \in [T] \tag{12}$$

Theorem 1: Problem $(\mathcal{O})$ *is strongly NP-hard.*

Proof: We defer the proof to Appendix A.

B. Relaxed Formulation

Note that the coflow-level constraints Eq. (2)-(12) of the original formulation $(\mathcal{O})$ are complicated nonlinear formulas with infinite variables, which make the original formulation hard to analyze and even harder to approximate. So we relax the formulation $(\mathcal{O})$ to a simple stage-level linear programming (LP). For simplicity, we defer the detailed derivation of LP to Appendix B.

Before presenting the relaxed LP, we first define some stage-level notations. Let $M_{g,k}$ denote the *k*th stage of cojob *g*, and let $\mathbb{M} = \{M_{g,k} | g \in \mathbb{G}, k \in [1, K_g]\}$ denote the stage set of all cojobs. The load of $M_{g,k}$ at port *p* is denoted as $L_p^{g,k}$. Then, the relaxed LP is shown as the following Eq. (13)-(15) (**LP-Primal**).

$$(\textbf{LP-Primal}) \qquad \min \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} S_{g,k} \tag{13}$$

$$\text{Subject to: } S_{g,k} - S_{g,k-1} \geq L_p^{g,k},$$
$$\forall g \in \mathbb{G}, k \in \left[K_g\right], p \in \mathbb{P} \tag{14}$$

$$\sum_{M_{g,k}\in\boldsymbol{M}} L_p^{g,k} S_{g,k} \geq f_p(\boldsymbol{M}),\ \forall p \in \mathbb{P}, k \in \left[1, K_g\right], \boldsymbol{M} \subseteq \mathbb{M} \tag{15}$$

In the **LP-Primal**, the first constraint Eq. (14) is derived by relaxing the coflow-level constraints. As the bandwidth capacity is normalized to 1, $L_p^{g,k}$ in Eq. (14) also denotes the minimum data transfer time of $M_{g,k}$ at port p. Thus constraint Eq. (14) means that the completion time of a stage must be at least $L_p^{g,k}$ time units later than the completion time of the previous stage within the same cojob. The second constraint Eq. (15) is inspired by Queyranne [29], which is well known to be an LP relaxation for the concurrent open shop problem and is widely applied in parallel scheduling problems like parallel machine scheduling [30] and coflow scheduling [13], [31]. In Eq. (15), $f_p(\boldsymbol{M})$ is derived by

$$f_p(\boldsymbol{M}) = \frac{1}{2}\left(\sum_{M_{g,k}\in\boldsymbol{M}} \left(L_p^{g,k}\right)^2 + \left(\sum_{M_{g,k}\in\boldsymbol{M}} L_p^{g,k}\right)^2\right). \tag{16}$$

Lemma 1: The optimal schedule of problem (**LP-Primal**) is a lower bound of the optimal schedule of problem $(\mathcal{O})$. For any optimal schedule OPT for problem $(\mathcal{O})$, let$(S_{g,k}^{OPT})_{g\in\mathbb{G},k\in[K_g]}$

to be the stage completion times under the schedule of OPT. Accordingly, let $(S^{LP}_{g,k})_{g\in\mathbb{G},k\in[K_g]}$ to be the stage completion times under the optimal schedule LP for problem (**LP-Primal**). Then, $\sum_{g\in\mathbb{G}}\sum_{k\in[K_g]} S^{LP}_{g,k} \le \sum_{g\in\mathbb{G}}\sum_{k\in[K_g]} S^{OPT}_{g,k}$.

Proof: We defer the proof to Appendix C.

Note that the problem (**LP-Primal**) is close to the problem of parallel dedicated machine scheduling with chain-precedence constraints (denoted as $PD|chains|\sum f_j(C_j)$) [32], [33], [34]. The problem $PD|chains|\sum f_j(C_j)$ is proved to be NP-Hard when there are more than three chains [34] and few works aimed at this problem in the past. To the best of our knowledge, there are only approximation algorithms for this problem dealing with quite special cases, e.g., the cases with at most two dependency chains or cases with at most two machines [32], [33], [34]. However, our problem has more dependency chains (e.g., a number of cojobs and the stages of a cojob are a chain) and more coupled resources (e.g., a flow leaves from the source port and arrives at a destination port). Based on the above investigation, to approximately solve the problem, we propose a permutation-schedule method composed of two steps. The first step is to permute stages of cojobs by using a simple primal-dual algorithm (see Section III-C) and the second step is to schedule stages according to the permutation (see Section III-D).

C. Permute Cojob Stages.

$$(\textbf{Dual}) \quad \max \sum_{p\in\mathbb{P}}\sum_{g\in\mathbb{G}}\sum_{k\in[K_g]} \alpha_{p,g,k} L^{g,k}_p + \sum_{p\in\mathbb{P}}\sum_{\boldsymbol{M}\subseteq\mathbb{M}} \beta_{p,\boldsymbol{M}} f_p(\boldsymbol{M}) \tag{17}$$

$$\text{Subject to}: \sum_{p\in\mathbb{P}}\left(\alpha_{p,g,k}-\alpha_{p,g,k+1}\right) + \sum_{p\in\mathbb{P}}\sum_{\boldsymbol{M}\ni M_{g,k}} L^{g,k}_p \beta_{p,\boldsymbol{M}} \le 1,$$

$$\forall g\in\mathbb{G}, k\in\left[K_g\right] \tag{18}$$

$$\alpha_{p,g,k}\ge 0,\ \forall p\in\mathbb{P}, k\in\left[K_g\right] \tag{19}$$

$$\beta_{p,\boldsymbol{M}}\ge 0,\ \forall p\in\mathbb{P}, \boldsymbol{M}\subseteq\mathbb{M} \tag{20}$$

The primal-dual algorithm (PDA) to permute stages is shown in *Algorithm 1*. PDA first initializes the unscheduled stage set $\mathbb{M}$ to containing all stages and then initializes all the dual variables to 0 (*line 1-2*). The precedence relations among stages in the same cojob are broken by fixing the dual variable $\alpha_{p,g,k}$ (*line 3-7*). Then PDA assigns each stage a weight (*line 9-10*) according to the dual constraints (18). And the weight is used to determine the stage permutation order. More specifically, PDA permutes stages in the reverse order iteratively (*line 13-22*). In any iteration, PDA chooses the port with the heaviest load caused by unscheduled stages (*line 15*), observes the ratios of each unscheduled stage's weight to its load on the chosen port, and picks the stage with the minimum ratio to be scheduled in the last (*line 16-17*). Then PDA

Algorithm 1 Primal-Dual Algorithm (PDA) to Permute Stages

Require:
port set $\mathbb{P}=\{1,2,\ldots,2P\}$; cojob set $\mathbb{G}=\{1,2,\ldots,G\}$; for any cojob $g\in\mathbb{G}$: number of stages K_g, the load at port p caused by kth stage $L^{g,k}_p$, for all $p\in\mathbb{P}$ and $k\in[K_g]$; Total stage number $l=\sum_{g\in\mathbb{G}}K_g$.

Ensure:
Stage permutation order $\sigma(1),\sigma(2),\sigma(3),\ldots,\sigma(l)$;

1: Initialize unscheduled stage set $\mathbb{M}=\{M_{g,k}|g\in\mathbb{G},k\in[K_g]\}$;
2: Initialize $\alpha_{p,g,k}=0$ and $\beta_{p,\boldsymbol{M}}=0,\forall p\in\mathbb{P},g\in\mathbb{G},\boldsymbol{M}\subseteq\mathbb{M}$;
3: **for** each g in $\mathbb{G}$ **do**
4: **for** $k=1,2,\ldots,K_g$ **do**
5: $\gamma(g,k)=\arg\max_{p\in\mathbb{P}} L^{g,k}_p$
6: $\alpha_{\gamma(g,k),g,k}=1-(\frac{1}{2})^k$
7: $\alpha_{p,g,k}=0,\forall p\neq\gamma(g,k)$
8: **end for**
9: **for** $k=1,2,\ldots,K_g$ **do**
10: $w_{g,k}=1+\sum_{p\in\mathbb{P}}\alpha_{p,g,k+1}-\sum_{p\in\mathbb{P}}\alpha_{p,g,k}$
11: **end for**
12: **end for**
13: $L_p=\sum_{g\in\mathbb{G}}\sum_{k\in[K_g]}L^{g,k}_p,\ \forall p\in\mathbb{P}$ ▷load on port i
14: **for** $i=l,l-1,\ldots,l$ **do**
15: $\lambda(i)=\arg\max_{p\in\mathbb{P}}L_p$ ▷most bottlenecked port
16: $g',k'=\arg\min_{g,k}\frac{L^{g,k}_{\lambda(i)}}{w_{g,k}},\ \forall M_{g,k}\in\mathbb{M}$
17: $\sigma(i)\leftarrow M_{g',k'}$
18: $\rho(i)=w_{g',k'}/L^{g',k'}_{\lambda(i)}$
19: $w_{g,k}=w_{g,k}-\rho(i)L^{g,k}_{\lambda(i)},\ \forall M_{g,k}\in\mathbb{M}$ ▷adjust weights
20: $\beta_{\lambda(i),\mathbb{M}}=\rho(i)$
21: $L_p=L_p-L^{g',k'}_p,\ \forall p\in\mathbb{P}$ ▷update port loads
22: $\mathbb{M}\leftarrow\mathbb{M}\backslash M_{g',k'}$
23: **end for**
24: **return** $\sigma(1),\sigma(2),\sigma(3),\ldots,\sigma(l)$.

adjusts the weights of unscheduled stages to ensure the feasibility of dual constraints (*line 18-19*). Finally, PDA raises the dual variable $\beta_{\mu(i),\mathbb{M}}$ until the dual constraint for the chosen port and set $\mathbb{M}$ becomes tight (*line 20*). The loads of each port and set $\mathbb{M}$ are also updated for next iteration (*line 21-22*).

D. Order-Preserving Scheduling

High-level idea: Given a stage permutation order produced by Algorithm PDA, at any time, the flows from stage $\sigma(k)$ are able to use the bandwidth of either the ingress or egress if and only if flows with higher stage permutation orders have stopped using the bandwidth of these ports. Such an order-preserving scheduling method is easily realized by using the existing priority-enabled network (see the implementation of Grouper in Section IV). Note that the order of cojob-stages is produced by considering the precedence constraints of stages and ignoring the interdependencies of coflows (*line 3-7*). On one hand, the produced order can satisfy the precedence constraints of stages in the same cojob. On the other hand, the coflows within a cojob-stage can be scheduled in order of their generation. Then the transmission demands and precedence constraints of coflows (i.e., Eq. (2)-(9)) can be satisfied.

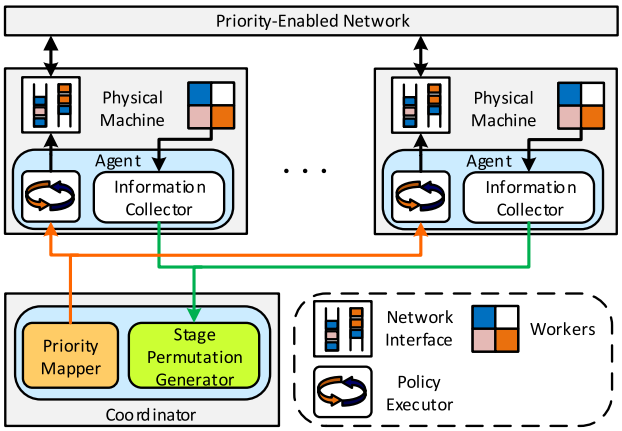


Fig. 6. System overview of Grouper.

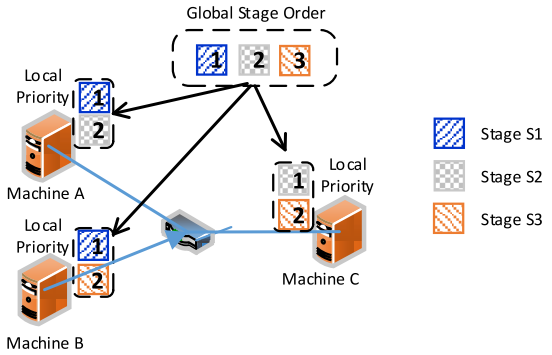


Fig. 7. An illustration of mapping the global order to the local priority.

In addition, our order-preserving scheduling naturally satisfies the constraints of link capacity and non-negative flow-rate constraints (i.e., Eq. (10)-(12)). Therefore, the solution obtained by our scheduling method satisfies all constraints of the original problem and thus is feasible for the original problem. Moreover, it is sufficient to minimize average cojob stage completion times (SCT). In the following, we show that it achieves average cojob SCT within 2x of optimal value for problem $(\mathcal{O})$.

Lemma 2: Let $\boldsymbol{M}(i) = \{\sigma(1), \sigma(2), \ldots, \sigma(i)\}$ denote the set of unscheduled stages at the beginning of iteration i. The following properties hold.

(a) $\beta_{p,\boldsymbol{M}(i)} = 0$ for all $p \neq \lambda(i)$;

(b) Nonzero $\beta_{p,\boldsymbol{M}}$ can be written as $\beta_{\lambda(i),\boldsymbol{M}(i)}$ for all $i \in [l]$;

(c) The dual constraints Eq. (18) are satisfied with equality;

(d) The dual constraints Eq. (19)(20) are satisfied.

Proof: We defer the proof to Appendix D.

Theorem 2: Given a set of DL cojobs $\mathbb{G}$, all cojobs arrive at time 0. Any order-preserving and per-flow rate allocation algorithm that schedules flows in the order generated by PDA, achieves cojob SCT within 2 $\times$ of the optimal average cojob SCT for the original problem (O).

Proof: We defer the proof to Appendix E.

IV. Implementation of Grouper

In this section, we present how to realize the above mentioned scheduling method in a system by using the existing priority-enabled network. The method is implemented in a real system, Grouper, and the overview of Grouper is presented in Section IV-A. The realizing of order-preserving scheduling over the existing network is presented in Section IV-B. Some practical considerations are presented in Section IV-C.

A. System Overview

Grouper is implemented in Python using about 1500 lines of code, which can be downloaded from Github (https://github.com/willzhoupan/Grouper). As shown in Fig. 6, Grouper consists of a centralized coordinator and an agent at each physical machine. The coordinator periodically asks the agents to collect cojob information (such as the cojob ID, job ID, interactions between PSs and workers) from log files. After receiving information from the agents, the coordinator generates a stage permutation by using algorithm PDA and assigns priorities to different stages according to their orders in the permutation (see Section IV-B). Then the coordinator sends the priorities of stages to agents that are responsible for the execution of the order scheduling. For any given flow, the agent offloads it into corresponding priority queues according to its stage priority (see Section IV-B).

B. Grouper Over Existing Network

If the number of priority queues of the underlying network is infinite, the implement of Grouper is simple: for any given flow, the agent just offloads it into the queue whose priority is equal to the order of the stage it belongs to. In practice, the underlying network fabric only supports limited priority queues due to hardware limitations. For example, Edge-core AS4610-54T switch adopted by our cluster only supports 8 priority queues. Moreover, in most datacenters or clusters, some of the queues are reserved for other purposes [35] (e.g., fault tolerance). Thus, there is a small number of queues that are available for Grouper.

With finite number q of queues, the implementation of Grouper approximates the ideal performance by mapping the stages' permutation orders into local priorities. More specifically, after generating a stage permutation (or global order), for each machine, the coordinator assigns priorities to the stages at this machine in the order of the permutation. For example, as shown in Fig. 7, a three-server cluster hosts three stages belonging to different cojobs. Machine *C* hosts stage *S*2 and *S*3, the coordinator maps the global order 2 and 3 to local priority 1 and 2 for stage *S*2 and *S*3, respectively. Similarly, for machine *B*, the coordinator maps global order <1,3> to local priority <1,2>. For machine *A*, the coordinator maps global order <1,2> to local priority <1,2>.

With the stage priorities, agents offload flows of stages to corresponding priority queues. To realize our order-preserving scheduling method, both the ingress and egress ports should support multiple priority queues. However, no tool alone in existing operating systems (i.e., Linux) or switches can support multiple queues for both ingress and egress ports of the network. Thus Grouper combines the queues of Linux

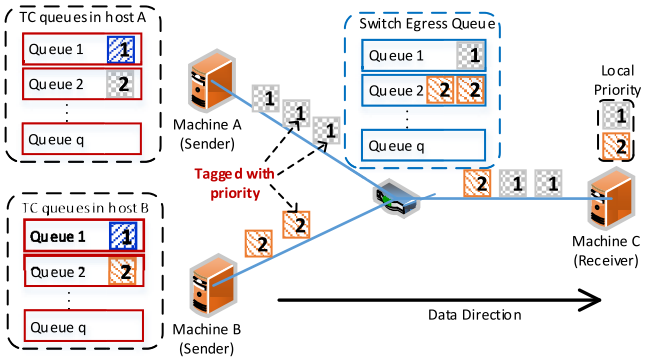


Fig. 8. An illustration of scheduling flows using existing network fabric.

TC (traffic controller) and queues of switches to realize our method.

For ingress ports of the fabric, Linux TC is adopted to create multiple priority queues for sending flows to the network. As shown in Fig. 8, machine *A* and machine *B* at the left side send data to machine *C* at the right side. Machine *A* and machine *B* just simply assign flows into their TC queues according to their stage priorities (the priorities are shown in Fig. 7). Meanwhile, Grouper adopts the egress ports of the switches to provide multiple queues for flows output to the machines. As shown in Fig. 8, the flows from machine *A* and *B* to *C* are mapped into the switch egress queue connecting to machine *C*. A flow's priority at the switch is equal to the local priority of its stage at the receiver side (see machine *C*). To realize this, Grouper agents of the senders use IP-tables, a Linux kernel tool, to modify the DSCP field of IP header [36]. Then packets of the flows are tagged with their priorities at the egress ports, as shown in Fig. 8. Note that if the priority is larger than q, we map the flows into queue q.

Through above mapping from global orders to local priorities, at any given ingress/egress port, the cojob-stage with the highest order will be assigned into the highest priority queue, and the other cojob-stages with lower orders will be assigned into lower priority queues, thus the cojob-stage with the highest order is scheduled before the other cojob-stages. Once the cojob-stage with the highest order is completed, the cojob-stage with the second-highest order will be assigned to the highest priority queue. As a result, the cojob-stages are completed in their orders. Extensive experiments and simulations of researches have shown that using finite local priority queues can achieve near ideal performance of using infinite priority queues [37], [38].

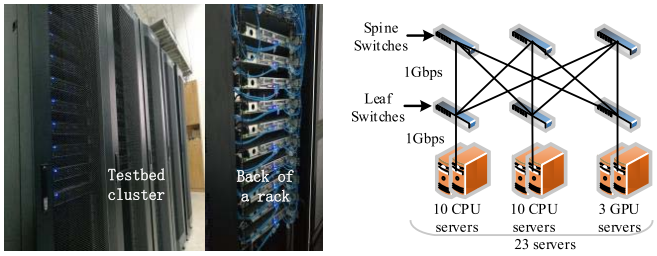
C. Practical Considerations

From offline to online: There exist some state-of-the-art frameworks that can transform an offline-algorithm to an online-algorithm with theoretical guarantees [39], [40], [41]. In these frameworks, the timeline is divided into exponentially increasing sized intervals. At the beginning of each interval, an $(\alpha, \beta)-$*approximation* algorithm is used to select released and unscheduled jobs, then a $\gamma-$*approximation* offline algorithm can be used to schedule the selected set of jobs. These offline-to-online transformations require the exact duration time of each job and can achieve $(2\alpha\beta+\gamma)-$*competitive*. However, in hyperparameter searching process, the duration time of each job is not known in advance, since ML practitioners will kill a portion of jobs with poor performance at the end of each cojob-stages, the achieved performance of a job is not known in advance and thus when the job would be killed is not known yet. As a result, these state-of-art frameworks cannot be applied to our work. In addition, when the time interval divided by these frameworks becomes too large, if there are some stages leaving or arriving during the time interval, these frameworks cannot react to the leaving and arriving of stages, and some stages that should be prior scheduled are assigned into low priority queues. Thus, it seems better to run the offline algorithm, reorder the stages and update corresponding priorities of stages upon a stage is arriving or leaving. While, in large clusters, the cojob-stages may be frequently arriving and leaving, which requires frequently update the priorities of cojob-stages. This can cause a high overhead for the scheduler and may cause instability for the cluster fabric. Based on the above considerations, we seek a trade-off between the performance and theoretical guarantees by running the offline-algorithm at a fixed time interval.

The offline algorithm PDA assumes that the total stage number or iteration number of a given job is known prior. In practice, the stage number of a job is not known until it is stopped. Thus, in the implementation of Grouper, we assume that all jobs in a cojob can be run to completion (or be able to finish all stages). Then the coordinator can run algorithm PDA periodically. Then the coordinator maps the obtained global stage orders into local orders, which are sent to the distributed agents to perform flow scheduling. Though this transformation may lose some performance compared with the offline-algorithm, it is practical in reality and easy to deploy. And our testbed experiments and simulations show that it significantly outperforms advanced network designs (see the next section). The mathematical analysis of the online transformation is not presented in this paper, but it will be an important future work.

Work conservation and starvation: As the priority queues of the underlying network are work-conserving, Grouper is naturally work-conserving. In addition, the priority queues of the underlying network can support many work-conserving algorithms to schedule packets in different queues, such as strict priority (SP), Weighted Round Robin (WRR). This can provide many choices for Grouper to achieve different goals. For example, we can configure WRR for the queues to avoid starvations.

Co-exist with other flows: Despite the flows generated by model synchronization among PSs and workers, every DL framework has some latency-sensitive messages, such as control messages and heartbeat messages. Moreover, the management of the cluster also generates some short messages requiring low latency. To guarantee the latency of these messages, our implementation of Grouper reserves the highest priority 0 to these messages.



(a) (b)

Fig. 9. Testbed. (a) Our testbed cluster; (b) The testbed topology.

V. Evaluation and Analysis

A. Evaluation Settings

Testbed: We build a testbed containing 20 CPU-servers and 3 GPU-servers (see Fig. 9(a)). The servers are organized across three racks and are interconnected by a leaf-spine network topology which containing 3 leaf (or top of rack) switches and 3 spine switches, as shown in Fig. 9(b). Both the leaf and spine switches are 48-port Edge-core AS4610-54T 1GbE switches and every switch port support 8 priority queues. Moreover, each GPU server has one Intel E5-2650 CPU, two NVIDIA 1080Ti GPUs, 64GB memory, one 500GB SSD, and one 4TB HDD. Each CPU server is equipped with two Intel E5-2609 CPUs, 32GB memory, and two 300GB HDDs. Tensorflow is adopted as the training framework and Kubernetes 1.7 is deployed to schedule submitted training jobs.

Simulator: To evaluate the performance of Grouper in large-scale clusters as well as its performance with sensitive parameter settings, we realize a discrete-time simulator based on NS-3 platform [42], a widely used network simulator. We realize two kinds of applications for network fabric built by NS-3: a) the application that mimics the data transmissions of distributed training jobs and b) the application that generates the scheduling priorities of training jobs. Meanwhile, we adopt job traces collected from our cluster as the input to our simulator. The job traces contain the following information: job configurations (e.g., cojob id, job id, model size, worker number, and server number), training loss varying with iteration, accuracy varying with iteration, number of training stages. The source code of our simulator is available at Github (https://github.com/willzhoupan/Grouper).

Baselines: We compare Grouper with another three advanced network designs: (i) Baraat [23], which schedules flows of a task as a whole and schedules different tasks in a FIFO manner. In our implementation of Baraat, we define each stage as a task and thus Baraat schedules different stages in a FIFO manner; (ii) Sincronia [13], which leverage the priority queues of the underlying network to schedule coflows. However, it aims to minimize average coflow completion time. (iii) Per-flow fair share (FS), which is widely adopted by existing datacenter transport mechanisms, such as DCTCP, TCP.

Metrics: We adopt the following four metrics as indicators of our system performance: (i) the average stage completion time (SCT), a smaller SCT indicates that more stages are completed per unit time; (ii) average job completion time (JCT), a smaller JCT indicates that more unpromising jobs are stopped per unit time; (iii) the average time to reach 90% of the converged accuracy, indicating how fast a job can achieve a high training accuracy; (iv) average stage slowdown (ASSD). For any given stage, we define its slowdown as the ratio between its completion time and its minimum completion time if running alone in the network, shown as follows:

$$\text{Stage slowdown} = \frac{\text{Compared SCT}}{\text{Minimum SCT if running alone}}.$$

The above definition implies that a smaller slowdown achieves smaller SCT and thus more stages are completed.

TABLE III
Models and Datasets Used for Experiments and Simulations

Models	Size (MB)	Dataset	Application
DeepSpeech2	160	LibriSpeech	Speech recognition
ResNet152	230	ImageNet	Image classification
AlexNet	250	ImageNet	Image classification
VGG19	580	ImageNet	Image classification

B. Testbed Experiments

Workload: In our testbed experiments, we train the most popular models for image classification and speech recognition (see Table III) to produce workload in our cluster. For each model, we launch a cojob with 8 training jobs at a time and adopt Successful-Halving method to evaluate these jobs. Thus each cojob has 4 stages in total. The 4 stages are set to containing 500, 1000, 2000, and 4000 iterations, respectively, and this setting is sufficient to train a model to achieve relatively high accuracy in our experiment and is sufficient to identify the performance of different hyperparameter configurations. For any given cojob, when the third stage is completed, only one job in it is considered as promising and can be run to completion. Then we will launch another cojob for its model. And the candidate hyper-parameter configurations of the new cojob are produced by using random search [17]. Due to the limit scale of our cluster, we configure 2 workers and 2 servers for each job. Furthermore, since the model sizes are very large, we downsize the training dataset for faster convergence. After downsizing, we can finish each experiment within 72 hours and we repeat each experiment for 3 times to obtain the average results.

Performance improvement: The CDF in Fig. 10(a) shows that Grouper achieves smaller stage slowdown overall, in comparison with Baraat, Sincrona, and FS. Compared with the baseline network designs, Grouper can reduce 90-th percentile stage slowdown by up to 14.6%, 34.1%, 27.5%, respectively. Moreover, the corresponding improvements in average stage slowdown (ASSD) are 22.7%, 26.8%, and 34.7%, as shown in Fig. 10(b).

Characteristics of completion times in Fig. 11 imply that Grouper can significantly improve training efficiency. Fig. 11(a) shows that Grouper reduces average stage completion time (SCT) by up to 22.5%, 24.2%, 31.0%, compared with Baraat, Sincrona, and FS, respectively. In terms of average cojob completion times (CoJCT), improvements of Grouper are 37.1% and 44.0%, compared with Baraat and FS. Note

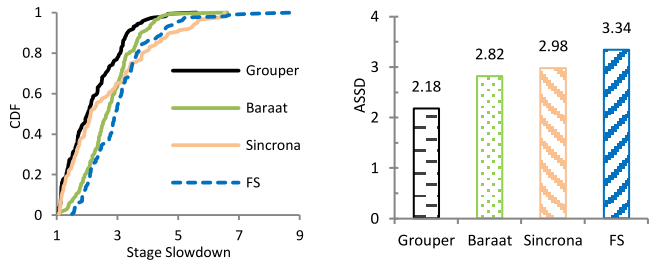


Fig. 10. [testbed] Statistics of stage slowdown. (a) CDF of stage slowdown; (b) Average stage slowdown. Grouper outperforms the other three network designs in terms of the performance of stage slowdown.

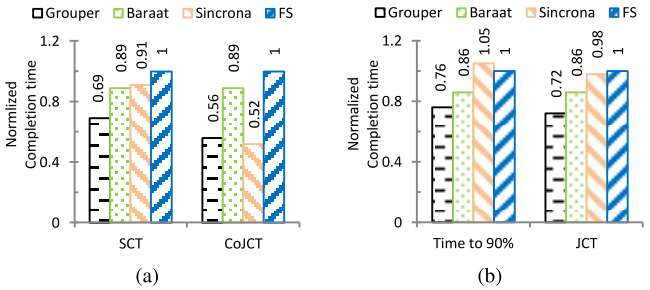


Fig. 11. [testbed] Characteristics of completion times. (a) Average stage completion time (SCT) and average cojob completion time (CoJCT); (b) Job completion time (JCT) and time to achieve 90% of the converged training accuracy. All the times are normalized to FS since FS is widely adopted by current transport protocols in datacenters (such as TCP and DCTCP).

that CoJCT of Sincrona is close to that of Grouper. This is because that Sincrona greedily schedules coflows of smaller models, which is noneffective to reduce SCT.

Fig. 11(b) shows that Grouper can reduce times to achieve 90% of converged training accuracy by 11.6%, 27.6%, and 24.0%, in comparison with Baraat, Sincrona, and FS, respectively. This implies that Grouper can help training jobs quickly achieve high training accuracies. Moreover, Grouper can also reduce average job completion times (JCT) by up to 16.3%, 26.5%, and 28.0%, compared with Baraat, Sincrona, and FS, respectively. This implies that more jobs with unpromising hyperparameter configurations are killed per unit time when Grouper is adopted, which is helpful for accelerating hyperparameter searching.

Performance insight: To reveal why the baseline network designs perform poorly than Grouper, we show the average stage completion times of models with various sizes in Fig. 12.

For large models (such as VGG19), the average SCT of Baraat is much smaller than that of FS. While, for small models (such as DeepSpeech2), the average SCT of Baraat is similar to that of FS. The reason is that Baraat schedules stages like a FIFO-manner, so that a portion of stages of small-model cojobs are blocked by stages of large-model cojobs. As the later stages of large-model cojobs are much longer than the early stages of small-model cojobs, FIFO-manner is suboptimal for minimizing average SCT.

For small models (such as DeepSpeech2), the average SCT of Sincrona is much smaller than that of FS. While, for

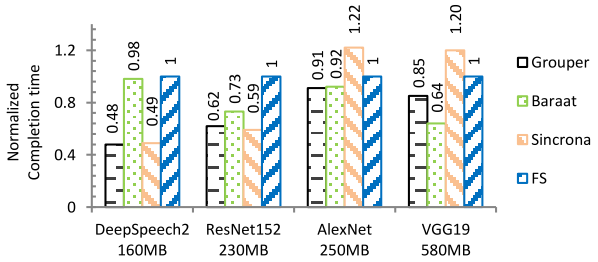


Fig. 12. Average SCT of models with different sizes when using different network designs. All the times are normalized to FS.

large models (such as VGG19), the average SCT of Sincrona is significantly higher than that of FS. This is because that Sincrona greedily schedules flows from small-model jobs. As later stages of small-model jobs are much longer than the early stages of large-model jobs, greedily scheduling stages of small-model jobs can lead to suboptimal performance for minimizing average SCT.

FS employs max-min fairness for flows at bottleneck links, which is blind to stage sizes. The shorter stages cannot finish as soon as possible, which is also suboptimal for minimizing average SCT. On the contrary, Grouper can prior schedule shorter stages at the bottleneck link, which is effective to reduce average SCT.

Discussion: In large industrial clusters, there exist more kinds of training cojobs with various models and datasets, and these cojobs belong to various ML practitioners with different ML experience. Due to the limited resources and users, the job settings in our experimental cluster are not the same as the job settings in real large industrial clusters. But our experimental result can denote the usefulness of our method in large industrial clusters. The reason is that our method can permute and schedule these cojob-stages by considering both the workload at each ingress/egress port of the cluster fabric and the stage-level information (e.g., stage sizes and data transmission requirements of a stage at each ingress/egress port). Different settings of jobs and different workloads of clusters can result in different degrees of performance improvement, but the effectiveness and usefulness of our method are not changed.

C. Sensitive Analysis

Simulation setup: In our simulations, we build a cluster containing 60 GPU-servers and every GPU-server contains 4 GPUs. As shown in Fig. 13(a), we adopt a leaf-spine network topology to interconnect the GPU servers. The servers are organized across 6 racks and each rack holds 10 GPU servers attaching to a leaf (or top of rack) switch. The leaf switches are connected to 6 spine switches. Note that all the links have 10Gbps capacity.

Varying workloads: The performance improvement of Grouper is highly related to the cluster workload. When the workload becomes heavier, the network fabric is more congested and it becomes more necessary to schedule flows generated by training jobs. In the following simulations, the job arrival interval follows the distribution of Microsoft's deep

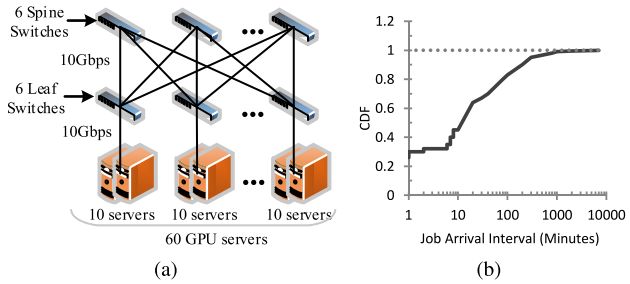


Fig. 13. Simulation Setup: (a) The cluster topology containing sixty GPU servers and twelve 10Gbps switches; (b) Distribution of job arrival intervals.

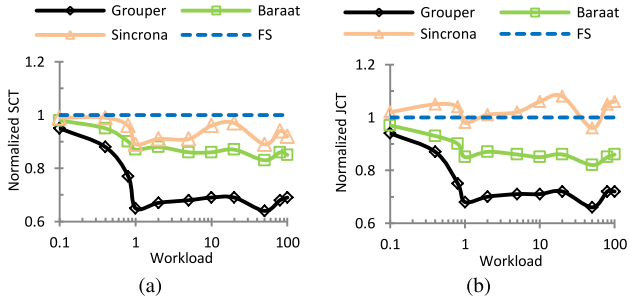


Fig. 14. Simulation performance under different workloads. (a) Average SCT varying with workload; (b) Average JCT varying with workload. All the completion times are normalized to FS.

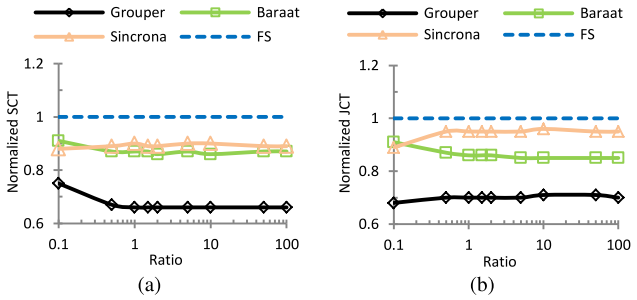


Fig. 15. Simulation performance under different ratio of communication time to computation time. (a) Average SCT varying with ratio; (b) Average JCT varying with ratio.

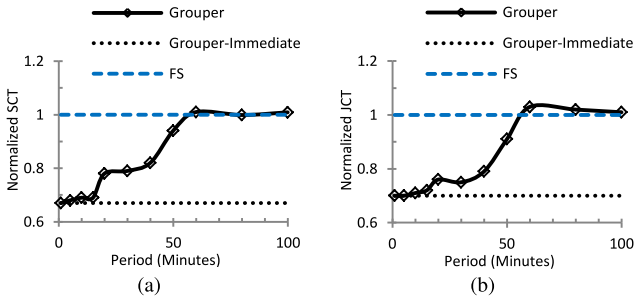


Fig. 16. Simulation performance under different scheduling periods: (a) Average SCT varying with periods; (b) Average JCT varying with periods. All completion times are normalized to FS.

learning cluster [7], [43], as shown in Fig. 13(b). When a job arrives, we randomly choose a job in our history traces collected from our testbed cluster as an input to the simulator. We generate different workload in the cluster by varying job arrival intervals. For example, we generate 2.0x workload by compressing 50% of the scale of the horizontal axis in Fig. 13(b); we generate 0.5x workload by expanding 2.0x of the scale of the horizontal axis. By adopting this method, we generate different workloads to evaluate Grouper, Baraat, Sincrona, and FS, in terms of average SCT and JCT.

Fig. 14 shows that Grouper can significantly reduce average SCT and JCT in general, in comparison with the other three network designs. More specifically, the performance improvements of Grouper increase as the workload increases from 0.1x to 1.0x. The reason is that the network fabric becomes more congested when the workload increases. While, when the workload is larger than 2.0x, the performance improvements of Grouper tend to be stable. This is because the resources of the cluster are limited, the number of running jobs in the cluster is also limited and most arrived jobs are waiting in the queue. Network scheduling cannot further improve performance.

Varying the ratio of communication time to computation time: Our original formulation ignores the computation time for simplicity. In practice, the computation time for small models is as large as communication time. In this simulation experiment, we evaluate the performance of Grouper by varying ratio of communication time to computation time, in terms of average SCT and JCT.

Fig. 15 shows that Grouper outperforms the other three network designs under different ratios. More specifically, Grouper can reduce average SCT by up to 24.1%, 25.9%, and 34.0%, in comparison with Baraat, Sincrona, and FS, respectively. The corresponding reductions in average JCT can be up to 17.6%, 26.3%, and 30.0%. Moreover, the performance improvements of Grouper tend to be stable when the ratio is larger than 0.5x. This implies that the performance improvement of Grouper is insensitive to the ratio of communication time to computation time for deep learning jobs.

Scalability of Grouper: Grouper periodically runs algorithm PDA and updates stage orders. As each stage of a cojob contains hundreds to thousands of iterations, the stage can last for tens of minutes or several hours. This means that the scheduling period of Grouper can be set to a long time.

Fig. 16 shows the performance degradation of Grouper when increasing the scheduling period from 1 minute to 100 minutes. More specifically, when the scheduling period is larger than 50 minutes, both the SCT and JCT of Grouper are close to those of FS. However, when the period is less than 20 minutes, the SCT and JCT of Grouper are stable and close to those of Grouper updating coflow order upon a stage starting or ending (see Grouper-Immediate in Fig. 16). This implies that the scheduling period of Grouper can be set to as long as 20 minutes, which hardly causes overhead for scheduling and thus results in good scalability for Grouper.

Gap between online and offline: In this simulation experiment, we evaluate the performance gap between the online version of Grouper (see Grouper in Fig. 17 and Fig. 18) and offline version of Grouper (see GrouperOffline in Fig. 17 and Fig. 18). Note that GrouperOffline knows all prior information including stages a job lasts for.

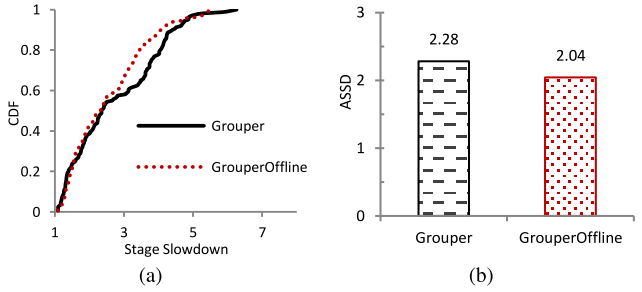


Fig. 17. Slowdown comparison between the online and offline version of Grouper: (a) CDF of stage slowdown ; (b) Average stage slowdown.

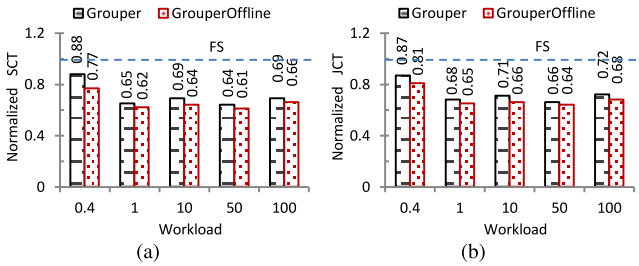


Fig. 18. Completion times of the online and offline version of Grouper. (a) Average SCT varying under different workload; (b) Average JCT under different workload. All the completion times are normalized to FS.

The CDF in Fig. 17(a) shows that the stage slowdown caused by GrouperOffline is slightly smaller than that caused by Grouper in general. The average stage slowdown of Grouper is increased by 12.0%, compared with GrouperOffline, as shown in Fig. 17(b). Fig. 18 shows that average SCT and JCT of Grouper is always slightly larger than those of GrouperOffline. Compared with GrouperOffline, the performance degradation of Grouper is at most 14.2%, in terms of average SCT and JCT. Such performance degradation is acceptable since there is a trade-off between theoretical performance and practical implementation. In the hyperparameter searching process, in a cojob, a portion of jobs with poor performance will be terminated at the end of each stage and the remaining jobs can continue in the next stage. Therefore, when a job would be terminated is not known in advance, and the real iteration number of a job is also not known in advance. However, in GrouperOffline, the real iteration number of a job is known in advance, which is not practical in reality. In Grouper, the remaining jobs are assumed to be able to finish all the stages. Such an assumption is not consistent with the fact and thus results in performance degradation, but it makes Grouper practical in reality. Moreover, though Grouper loses some performance compared with GrouperOffline, it still significantly outperforms state-of-art flow schedulers, as shown in the previous experiment and simulation.

VI. Related Work

Resource allocation in GPU clusters: Recently there have been many efforts on resource allocation in GPU clusters to achieve various goals. Optimus [44] dynamically adjusts the number of workers or PSs for DL jobs to improve resource utilization and job completion times. Harmony [45] adopts deep reinforcement learning method to learn interference between colocated DL jobs and optimize job placement in GPU clusters. SLAQ [46] builds resource-performance models for DL jobs and dynamically adjusts worker numbers for DL jobs to reduce the completion time of the early training stage during hyperparameter searching. Tiresias [43] and Gandiva [7] realize GPU time-sharing and schedule a cojob at a time to reducing SCTs for hyperparameter searching. All the above resource allocation systems are focused on the allocation of computation resources. While Grouper is focused on network scheduling. Thus the above researches are complementary to our work Grouper.

Network scheduling in datacenters: There have been many achievements to improve the performance of datacenter networks. Most of the works are aimed to minimize average flow completion time (FCT). For example, Eiffel [9] and PIAS [10] adopt multiple in-network prioritization to approximate the ideal of Shortest Remaining Processing Time First (SRPTF) for reducing FCT. However, minimizing FCT mismatches performance objectives of data-intensive applications in datacenters. Thus, coflow [47] is proposed to alleviate this mismatch and many achievements have emerged. For example, Varys [47] uses the smallest-effective-bottleneck-first (SEBF) heuristic to allocate sending rates of coflows at the source side. And Sincrona [13] offloads coflow scheduling to the priority queues of the underlying network. Almost all the works are aimed to minimize average coflow completion time (CCT). Meanwhile, Baraat [23] proposes another method, task-level scheduling, to alleviate the mentioned mismatch. Baraat takes the flows of a task as a whole and schedules flows from different tasks by using a FIFO manner, which can reduce task completion times. But all the mentioned works cannot reduce DL training time since their objectives (minimizing FCT or CCT) mismatch the performance objective (i.e., minimizing SCT) of searching hyperparameter for DL models mentioned in Section II. In contrast, Grouper can speed up the hyperparameter searching process by minimizing SCT of cojobs.

VII. Conclusion

In this paper, we show that minimizing SCT of cojobs can effectively accelerate the searching of hyperparameter settings. As the network communication is one of the major bottlenecks for distributed training, we formulate the problem of network scheduling as an optimization model and prove its NP-Hardness. To approximately solve the problem, we propose a permutation-and-schedule method, which adopts a well-designed primal-dual algorithm (PDA) to generate orders in which the flows of these training stages are scheduled. Our mathematical analysis shows that this method can achieve average SCT with 2x of the optimal. We also realize the method in a real system Grouper, which leverages the priority queues of the underlying network to schedule flows of different stages in the produced order. Our extensive testbed experiments and digital simulations show that Grouper outperforms the state-of-the-art network designs Baraat, Sincrona, and FS, in terms of performance metrics such as SCT, JCT and stage

slowdown. Note that algorithm PDA is offline, in the implementation of Grouper, we transform the method into an online manner by periodically running algorithm PDA and assuming all the left jobs can run to completion. Though there exists some performance degradation of the online algorithm compared with the offline one, the online algorithm is realizable. The mathematical analysis of Grouper's online modification is one of our future work. Moreover, our design assumes that the cluster is homogeneous, but the design of cojob scheduling that considers the heterogeneity of the cluster is interesting. In this case, stage time can be considered as an additional variable to design a more efficient scheduler. This is one of our important future work.

APPENDIX A
PROOF OF THEOREM 1

Proof: We reduce existing cojob-agnostic coflow scheduling problem (CACSP) to our problem $(\mathcal{O})$. The CACSP has been proven to be strongly NP-Hard [47]. Given any arbitrary instance of CACSP, we can construct a corresponding instance of $(\mathcal{O})$ as follows: assuming there are n coflows in CACSP, we construct n cojobs, each cojob containing one job with one coflow. Thus, the solution of the constructed instance is exactly the solution of the given CACSP instance. Therefore, if we could solve the constructed instance in polynomial time, we can also solve the SACSP instance in polynomial time. Since CACSP is strongly NP-Hard, problem $(\mathcal{O})$ is NP-Hard as well.

APPENDIX B
DERIVATION OF LP-PRIMAL

The derivation of the LP-Primal is shown as the following three steps:

(1) Relax coflow-level constraints to stage-level ones: By combining Eq. (3) and (7), we can obtain that

$$E_{g,j,k} = C_{g,j,I_{g,k}},\ \forall g \in \mathbb{G}, j \in \big[N_g\big], k \in \big[K_g\big] \tag{21}$$

By combining Eq. (2) and (21), we have that

$$S_{g,k} = \max_{j\in[N_g]} C_{g,j,I_{g,k}},\ \forall g \in \mathbb{G}, k \in \big[K_g\big] \tag{22}$$

Eq. (7), (8) and (22) indicate that

$$t^*_{g,j,i} \geq S_{g,k-1},\ \forall g \in \mathbb{G}, j \in \big[N_g\big], \\ k \in \big[K_g\big], i \in \big(I_{g,k-1}, I_{g,k}\big] \tag{23}$$

Eq. (23) means that stage k of cojob g starts after the completion of stage $k-1$ of cojob g. Thus, by combining Eq. (4)-(6), (22) and (23), we have that

$$\int_{t^*_{g,j,i}}^{F^{g,j,i}_{s,d}} r^{g,j,i}_{s,d}(t)dt = \int_{S_{g,k-1}}^{S_{g,k}} r^{g,j,i}_{s,d}(t)dt = d^{g,j}_{s,d} \tag{24}$$

Eq. (10) indicates that

$$\sum_{j\in[N_g]} \sum_{i\in[I_{g,max}]} \sum_{s\in\mathbb{I}} r^{g,j,i}_{s,d}(t) \leq 1,\ \forall g \in \mathbb{G}, d \in \mathbb{E}, t \in [T] \tag{25}$$

Now integrating the both sides of Eq (25) from $S_{g,k-1}$ to $S_{g,k}$, we can obtain that

$$\int_{S_{g,k-1}}^{S_{g,k}} \sum_{g\in[N_g]} \sum_{i\in[I_{g,max}]} \sum_{s\in\mathbb{I}} r^{g,j,i}_{s,d}(t)dt \leq \int_{S_{g,k-1}}^{S_{g,k}} 1 \cdot dt, \\ \forall g \in \mathbb{G}, k \in \big[K_g\big], d \in \mathbb{E}, t \in [T] \tag{26}$$

By combining Eq. (24), the left side of Eq. (26) can be rewritten as

$$\begin{aligned} &\int_{S_{g,k-1}}^{S_{g,k}} \sum_{g\in[N_g]} \sum_{i\in[I_{g,max}]} \sum_{s\in\mathbb{I}} r^{g,j,i}_{s,d}(t)dt \\ &= \sum_{j\in[N_g]} \sum_{i\in[I_{g,max}]} \sum_{s\in\mathbb{I}} \int_{S_{g,k-1}}^{S_{g,k}} r^{g,j,i}_{s,d}(t)dt \\ &= \sum_{j\in[N_g]} \sum_{i\in\left(I_{g,k-1},I_{g,k}\right]} \sum_{s\in\mathbb{I}} d^{g,j}_{s,d} \\ &= \sum_{j\in[N_g]} \sum_{s\in\mathbb{I}} \sum_{i\in\left(I_{g,k-1},I_{g,k}\right]} d^{g,j}_{s,d} \\ &= \sum_{j\in[N_g]} \sum_{s\in\mathbb{I}} \big(I_{g,k} - I_{g,k-1}\big)\, d^{g,j}_{s,d} \\ &= L^{g,k}_d \end{aligned} \tag{27}$$

where $L^{g,k}_d$ means the load of kth stage of cojob g at port d. The right side of Eq. (26) can be rewritten as

$$\int_{S_{g,k-1}}^{S_{g,k}} 1 \cdot dt = S_{g,k} - S_{g,k-1} \tag{28}$$

By combining Eq. (27) and (28), we have that

$$S_{g,k} - S_{g,k-1} \geq L^{g,k}_d,\ \forall g \in \mathbb{G}, k \in \big[K_g\big], d \in \mathbb{E} \tag{29}$$

Similarly, we have

$$S_{g,k} - S_{g,k-1} \geq L^{g,k}_s,\ \forall g \in \mathbb{G}, k \in \big[K_g\big], s \in \mathbb{I} \tag{30}$$

Eq. (9) and (10) can be simplified to

$$S_{g,k} - S_{g,k-1} \geq L^{g,k}_p,\ \forall g \in \mathbb{G}, k \in \big[K_g\big], p \in \mathbb{P} \tag{31}$$

(2) Introduce parallel inequalities: We define that the stage k of a given cojob g is denoted as $M_{g,k}$, and the stage set of all cojobs is denoted as $\mathbb{M} = \{M_{g,k} | g \in \mathbb{G}, k \in [1, K_g]\}$. For any subset $\boldsymbol{M} \subseteq \mathbb{M}$ and any feasible schedule, we assume that the stages in the subset $\boldsymbol{M}$ are ordered such that $S_{g(1),k(1)} < S_{g(2),k(2)} < \cdots < S_{g(|\boldsymbol{M}|),k(|\boldsymbol{M}|)}$, where $g(i) \in \mathbb{G}$ and $k(i) \in [1, K_{g(i)}]$. Then the nth stage in subset $\boldsymbol{M}$ could not be completed until all its previous stages are completed, thus we have that

$$S_{g(n),k(n)} \geq \sum_{m=1}^{n} L^{g(m),k(m)}_p,\ \forall p \in \mathbb{P}, n \in [1, |\boldsymbol{M}|] \tag{32}$$

Then, we get that

$$\sum_{n=1}^{|\boldsymbol{M}|} L^{g(n),k(n)}_p S_{g(n),k(n)}$$

$$\geq \sum_{n=1}^{|\boldsymbol{M}|} L_p^{g(n),k(n)} \sum_{m=1}^{n} L_p^{g(m),k(m)}$$
$$= \frac{\sum_{n=1}^{|\boldsymbol{M}|}\left(L_p^{g(n),k(n)}\right)^2 + \left(\sum_{n=1}^{|\boldsymbol{M}|} L_p^{g(n),k(n)}\right)^2}{2}$$
$$= f_p(\boldsymbol{M}),\ \forall p \in \mathbb{P}, k \in [1, |\boldsymbol{M}|] \tag{33}$$

Eq. (33) can be rewritten as follows:

$$\sum_{M_{g,k} \in \boldsymbol{M}} L_p^{g,k} S_{g,k} \geq f_p(\boldsymbol{M}),\ \forall p \in \mathbb{P}, k \in \left[1, K_g\right], \boldsymbol{M} \subseteq \mathbb{M} \tag{34}$$

(3) Relaxed LP: Finally, we get the following relaxed LP of the original formulation.

$$(\textbf{LP-Primal}) \qquad \min \sum_{g \in \mathbb{G}} \sum_{k \in [K_g]} S_{g,k}$$
$$\text{Subject to: (31), (34).} \tag{35}$$

APPENDIX C
PROOF OF LEMMA 1

Proof: To prove this, we map the feasible schedules for original problem ($\mathcal{O}$) to feasible solutions for (**LP-Primal**). We define that the stage k of a given cojob g is denoted as $M_{g,k}$, and the stage set of all cojobs is denoted as $\mathbb{M} = \{M_{g,k} | g \in \mathbb{G}, k \in [K_g]\}$. For any subset $\boldsymbol{M} \subseteq \mathbb{M}$ and any feasible schedule, we assume that the stages in subset $\boldsymbol{M}$ are ordered such that $S_{g(1),k(1)} < S_{g(2),k(2)} < \cdots < S_{g(|\boldsymbol{M}|),k(|\boldsymbol{M}|)}$, where $g(i) \in \mathbb{G}$ and $k(i) \in [K_{g(i)}]$. The nth stage in subset $\boldsymbol{M}$ could not be completed until all its previous stages are completed, thus we can obtain that

$$S_{g(n),k(n)} \geq \sum_{m=1}^{n} L_p^{g(m),k(m)},\ \forall p \in \mathbb{P}, n \in [1, |\boldsymbol{M}|]$$

Then, we get that

$$\sum_{n=1}^{|\boldsymbol{M}|} L_p^{g(n),k(n)} S_{g(n),k(n)} \geq \sum_{n=1}^{|\boldsymbol{M}|} L_p^{g(n),k(n)} \sum_{m=1}^{n} L_p^{g(m),k(m)}$$
$$= \frac{\sum_{n=1}^{|\boldsymbol{M}|}\left(L_p^{g(n),k(n)}\right)^2 + \left(\sum_{n=1}^{|\boldsymbol{M}|} L_p^{g(n),k(n)}\right)^2}{2}$$
$$= f_p(\boldsymbol{M}),\ \forall p \in \mathbb{P}, k \in [1, |\boldsymbol{M}|]$$

The above equation can be rewritten as follows:

$$\sum_{M_{g,k}} L_p^{g,k} S_{g,k} \geq f_p((\boldsymbol{M})),\quad \forall p \in \mathbb{P}, k \in \left[1, K_g\right], \boldsymbol{M} \subseteq \mathbb{M}$$

Thus, the constraint (34) of (**LP-Primal**) is satisfied. In addition, constraint (31) of (**LP-Primal**) is also satisfied since the schedule is feasible for problem ($\mathcal{O}$). This implies that $(S_{g,k}^{OPT})_{g \in \mathbb{G}, k \in [K_g]}$ is one of the feasible solutions for (**LP-Primal**), and clearly $\sum_{g \in \mathbb{G}} \sum_{k \in [K_g]} S_{g,k}^{LP}$ is at most the value of $\sum_{g \in \mathbb{G}} \sum_{k \in [K_g]} S_{g,k}^{OPT}$. Proof of Lemma 1 is completed.

APPENDIX D
PROOF OF LEMMA 2

Proof: For ease of notation, let $L_p^{\sigma(i)} = L_p^{g(k),k(i)}$ and $w_{\sigma(i)} = w_{g(i),k(i)}$. For algorithm PDA (*line 14-22*), let $w_{\sigma(i)}(n)$ denote the value of adjusted weight of stage $\sigma(i)$ at iteration $n = i, i+1, \ldots, l$. Let $\boldsymbol{M}(n) = \{\sigma(1), \sigma(2), \ldots, \sigma(n)\}$ denote the set of unscheduled stages at the beginning of iteration *n*.

Properties *(a)* and *(b)* can be directly obtained from algorithm PDA (*line 6-10,13-20*).

Property *(c)* is proved as follows: Algorithm PDA (*line 19-22*) implies that for any iteration $i = l-1, \ldots, 1$,

$$w_{\sigma(i)}(i) = w_{\sigma(i)} - \sum_{n=i}^{l} \rho(n) L_{\lambda(n)}^{\sigma(i)} \tag{36}$$

From *line 18*, we have that

$$\rho(i) = w_{\sigma(i)}(i+1) / L_{\lambda(i)}^{\sigma(i)} \tag{37}$$

By combining (*line 19*) and Eq. (37), we can obtain

$$w_{\sigma(i)}(i) = w_{\sigma(i)}(i+1) - \frac{w_{\sigma(i)}(i+1)}{L_{\lambda(i)}^{\sigma(i)}} L_{\lambda(i)}^{\sigma(i)} = 0 \tag{38}$$

By combining Eq. (36) and (38), we have

$$w_{\sigma(i)}(i) = w_{\sigma(i)} - \sum_{n=i}^{l} \rho(n) L_{\lambda(n)}^{\sigma(i)} = 0 \tag{39}$$

Thus

$$w_{\sigma(i)} = \sum_{n=i}^{l} \rho(n) L_{\lambda(n)}^{\sigma(i)},\ \forall i \in [l] \tag{40}$$

From *line 20*, we can obtain

$$\sum_{n=i}^{l} \rho(n) L_{\lambda(n)}^{\sigma(i)} = \sum_{n=i}^{l} \beta_{\lambda(n),\boldsymbol{M}(n)} L_{\lambda(n)}^{\sigma(i)} \tag{41}$$

From *line 10*, we have

$$w_{\sigma(i)} = 1 + \sum_{p \in \mathbb{P}} \alpha_{p,g(i),k(i)+1} - \sum_{p \in \mathbb{P}} \alpha_{p,g(i),k(i)} \tag{42}$$

By combining Eq. (42), (41) and (40), we can obtain that

$$1 + \sum_{p \in \mathbb{P}} \alpha_{p,g(i),k(i)+1} - \sum_{p \in \mathbb{P}} \alpha_{p,g(i),k(i)}$$
$$= \sum_{n=i}^{l} \beta_{\lambda(n),\boldsymbol{M}(n)} L_{\lambda(n)}^{\sigma(i)} \tag{43}$$

By applying property b to Eq. (43), we have that

$$1 + \sum_{p \in \mathbb{P}} \alpha_{p,g(i),k(i)+1} - \sum_{p \in \mathbb{P}} \alpha_{p,g(i),k(i)}$$
$$= \sum_{p \in \mathbb{P}} \sum_{\boldsymbol{M} \ni \sigma(i)} L_p^{\sigma(i)} \beta_{p,\boldsymbol{M}},\ \forall i \in [l] \tag{44}$$

And (44) can be rewritten as

$$\sum_{p \in \mathbb{P}} \left(\alpha_{p,g,k} - \alpha_{p,g,k+1}\right) + \sum_{p \in \mathbb{P}} \sum_{\boldsymbol{M} \ni M_{g,k}} L_p^{g,k} \beta_{\rho,\boldsymbol{M}} = 1,$$

$$\forall g \in \mathbb{G}, k \in \left[K_g\right] \tag{45}$$

Proof of property *(c)* is completed.

Property *(d)* is proved as follows:

From *line 6-7* of PDA, we can obtain that $\alpha_{p,g,k} \geq 0$ for all $p \in \mathbb{P}, g \in \mathbb{G}, k \in [K_g]$. Thus, constraints Eq. (19) are satisfied. From *line 6-10* of PDA, we have that

$$\begin{aligned} w_{g,k} &= 1 + \sum_{p\in\mathbb{P}} \left(\alpha_{p,g,k+1} - \alpha_{p,g,k}\right) \\ &= 1 + \sum_{p\in\mathbb{P}} \alpha_{p,g,k+1} - \sum_{p\in\mathbb{P}} \alpha_{p,g,k} \\ &= 1 + 1 - \left(\frac{1}{2}\right)^{k+1} - \left(1 - \left(\frac{1}{2}\right)^{k}\right) \\ &= 1 + \left(\frac{1}{2}\right)^{k+1} \end{aligned} \tag{46}$$

Eq. (46) implies that $w_{g,1} \geq w_{g,2} \geq \cdots \geq w_{g,K_g} \geq 0$, for all $g \in \mathbb{G}$. In addition, for PDA (*line 16-22*), at any iteration $i = 2, \ldots, l$, the choice of scheduled coflow $\sigma(i)$ at *line 16* implies that the adjusted weights $w_{\sigma(n)}(i) \geq 0$ for all $\sigma(n) \in \boldsymbol{M}(i)$. Thus $\beta_{\lambda(i),\boldsymbol{M}(i)} = \rho(i) \geq 0$, and $\beta_{p,\boldsymbol{M}} \geq 0$, for any $p \in \mathbb{P}, \boldsymbol{M} \subseteq \mathbb{M}$. The constraints Eq. (20) are satisfied. The proof of property *(d)* is completed.

APPENDIX E
PROOF OF THEOREM 2

Proof: Assuming that the total number of stages in $\mathbb{G}$ is l. Consider a stage order $\boldsymbol{\sigma} = \{\sigma(1), \sigma(2), \ldots, \sigma(l)\}$ produced by algorithm PDA, let $S_{g(i),k(i)}$, where $g(i) \in \mathbb{G}$, $k(i) \in [K_{g(i)}]$, denote the completion time of stage $\sigma(i)$. In addition, let $\boldsymbol{M}(i) = \{\sigma(1), \sigma(2), \ldots, \sigma(i)\}$ denote the set of unscheduled stages at the beginning of iteration i. For any order-preserving scheduler, completion times of stages satisfy that $S_{g(1),k(1)} \leq S_{g(2),k(2)} \leq \cdots \leq S_{g(l),k(l)}$, and $S_{g(i),k(i)} = \sum_{j\in[i]} L_{\lambda(i)}^{g(j),k(j)} = \sum_{j=1}^{i} L_{\lambda(i)}^{g(j),k(j)}$. For simplicity, let $S_{\sigma(i)} = S_{g(i),k(i)}$, $L_p^{\sigma(i)} = L_p^{g(i),k(i)}$ and $w_{\sigma(i)} = w_{g(i),k(i)}$. Thus, the objective value produced by the order-preserving scheduler is

$$\begin{aligned} &\sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} S_{g,k} \\ &\overset{\text{(i)}}{=} \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(\sum_{p\in\mathbb{P}} \left(\alpha_{p,g,k} - \alpha_{p,g,k+1}\right) + \sum_{p\in\mathbb{P}} \sum_{\boldsymbol{M}\ni M_{g,k}} L_p^{g,k} \beta_{p,\boldsymbol{M}} \right) S_{g,k} \\ &= \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(\sum_{p\in\mathbb{P}} \left(\alpha_{p,g,k} - \alpha_{p,g,k+1}\right) \right) S_{g,k} \\ &\quad + \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(\sum_{p\in\mathbb{P}} \sum_{\boldsymbol{M}\ni M_{g,k}} L_p^{g,k} \beta_{p,\boldsymbol{M}} \right) S_{g,k} \end{aligned} \tag{47}$$

where equation (i) holds by Lemma 2(c). The first item at the right side of Eq. (49) can be rewritten as:

$$\begin{aligned} &\sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(\sum_{p\in\mathbb{P}} \left(\alpha_{p,g,k} - \alpha_{p,g,k+1}\right) \right) S_{g,k} \\ &= \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(\sum_{p\in\mathbb{P}} \alpha_{p,g,k} - \sum_{p\in\mathbb{P}} \alpha_{p,g,k+1} \right) S_{g,k} \\ &\overset{\text{(ii)}}{=} \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(\left(1 - \left(\frac{1}{2}\right)^{k}\right) - \left(1 - \left(\frac{1}{2}\right)^{k+1}\right) \right) S_{g,k} \\ &= \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(-\left(\frac{1}{2}\right)^{k+1} \right) S_{g,k} \end{aligned} \tag{48}$$

where equation (ii) holds by PDA (*line 6-7*). By combining Eq. (47) and (48), we have that

$$\begin{aligned} &\sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} S_{g,k} \\ &= \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(-\left(\frac{1}{2}\right)^{k+1} \right) S_{g,k} \\ &\quad + \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(\sum_{p\in\mathbb{P}} \sum_{\boldsymbol{M}\ni M_{g,k}} L_p^{g,k} \beta_{p,\boldsymbol{M}} \right) S_{g,k} \end{aligned} \tag{49}$$

By moving the first item at the right side of Eq. 49 to the left side, we can obtain that

$$\begin{aligned} &\sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(1 + \left(\frac{1}{2}\right)^{k+1} \right) S_{g,k} \\ &= \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(\sum_{p\in\mathbb{P}} \sum_{\boldsymbol{M}\ni M_{g,k}} L_p^{g,k} \beta_{p,\boldsymbol{M}} \right) S_{g,k} \end{aligned} \tag{50}$$

And Eq. (50) implies that:

$$\begin{aligned} &\sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} S_{g,k} \\ &\leq \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(1 + \left(\frac{1}{2}\right)^{k+1} \right) S_{g,k} \\ &= \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \left(\sum_{p\in\mathbb{P}} \sum_{\boldsymbol{M}\ni M_{g,k}} L_p^{g,k} \beta_{p,\boldsymbol{M}} \right) S_{g,k} \\ &= \sum_{i\in[l]} \left(\sum_{p\in\mathbb{P}} \sum_{n\geq i} L_p^{\sigma(i)} \beta_{p,\boldsymbol{M}(n)} \right) S_{\sigma(i)} \\ &= \sum_{i\in[l]} \sum_{p\in\mathbb{P}} \sum_{n\geq i} L_p^{\sigma(i)} \beta_{p,\boldsymbol{M}(n)} S_{\sigma(i)} \\ &= \sum_{n\in[l]} \sum_{p\in\mathbb{P}} \sum_{i\leq n} L_p^{\sigma(i)} \beta_{p,\boldsymbol{M}(n)} S_{\sigma(i)} \\ &\overset{\text{(iii)}}{\leq} \sum_{n\in[l]} \sum_{p\in\mathbb{P}} \sum_{i\leq n} L_p^{\sigma(i)} \beta_{p,\boldsymbol{M}(n)} S_{\sigma(n)} \end{aligned}$$

$$
\begin{aligned}
&= \sum_{n\in[l]} S_{\sigma(n)} \sum_{p\in\mathbb{P}} \beta_{p,\boldsymbol{M}(n)} \sum_{i\le n} L_p^{\sigma(i)} \\
&\overset{\text{(iv)}}{=} \sum_{n\in[l]} S_{\sigma(n)} \beta_{\lambda(n),\boldsymbol{M}(n)} \sum_{i\le n} L_{\lambda(n)}^{\sigma(i)} \\
&\overset{\text{(v)}}{=} \sum_{n\in[l]} \left(\sum_{i\le n} L_{\lambda(n)}^{\sigma(i)} \right) \beta_{\lambda(n),\boldsymbol{M}(n)} \sum_{i\le n} L_{\lambda(n)}^{\sigma(i)} \\
&= \sum_{n\in[l]} \beta_{\lambda(n),\boldsymbol{M}(n)} \left(\sum_{i\le n} L_{\lambda(n)}^{\sigma(i)} \right)^2 \\
&\overset{\text{(vi)}}{\le} \sum_{n\in[l]} \beta_{\lambda(n),\boldsymbol{M}(n)} \left(2 - \frac{2}{l+1} \right) f_{\lambda(n)}(\boldsymbol{M}(n)) \\
&= \left(2 - \frac{2}{l+1} \right) \sum_{n\in[l]} \beta_{\lambda(n),\boldsymbol{M}(n)} f_{\lambda(n)}(\boldsymbol{M}(n)) \\
&\overset{\text{(vii)}}{=} \left(2 - \frac{2}{l+1} \right) \sum_{p\in\mathbb{P}} \sum_{\boldsymbol{M}\subseteq\mathbb{M}} \beta_{p,\boldsymbol{M}} f_p(\boldsymbol{M}) \\
&\le 2 \sum_{p\in\mathbb{P}} \sum_{\boldsymbol{M}\subseteq\mathbb{M}} \beta_{p,\boldsymbol{M}} f_p(\boldsymbol{M}) \\
&\le 2 \left(\sum_{p\in\mathbb{P}} \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} \alpha_{p,g,k} L_p^{g,k} + \sum_{p\in\mathbb{P}} \sum_{\boldsymbol{M}\subseteq\mathbb{M}} \beta_{p,\boldsymbol{M}} f_p(\boldsymbol{M}) \right) \\
&\overset{\text{(viii)}}{\le} 2 \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} S_{g,k}^{LP} \\
&\overset{\text{(ix)}}{\le} 2 \sum_{g\in\mathbb{G}} \sum_{k\in[K_g]} S_{g,k}^{OPT}
\end{aligned} \tag{51}
$$

where (iii) holds since stage $\sigma(n)$ is completed after all previous stages $(\sigma(i))_{i\le n}$ are completed and thus $S_{\sigma(i)} \le S_{\sigma(n)}$, for all $i \le n$; (iv) holds by Lemma 3.2(a); (v) holds since $S_{\sigma(n)} = \sum_{i\le n} L_{\lambda(n)}^{\sigma(i)}$ by using any work conserving rate allocation algorithm that schedules flows of stages with respect to the permutation order produced by PDA; (vi) holds by Lemma 3; (vii) holds by Lemma 2(b); (viii) holds since dual variables $\alpha_{p,g,k}$ and $\beta_{p,\boldsymbol{M}}$ are feasible for (**Dual**) (see Lemma 2(c)(d)); (ix) holds by Lemma 1. The proof of Theorem 2 is completed.

Lemma 3 [48]: For any $p \in \mathbb{P}$, and $\boldsymbol{M} \subseteq \mathbb{M}$, we have that $(\sum_{M_{g,k}\in\boldsymbol{M}} L_p^{g,k})^2 \le (2 - \frac{2}{l+1}) f_p(\boldsymbol{M})$.

REFERENCES

[1] K. Hazelwood *et al.*, "Applied machine learning at Facebook: A datacenter infrastructure perspective," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2018, pp. 620–629.

[2] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," 2019. [Online]. Available: arXiv:1901.05758.

[3] J. Rasley, Y. He, F. Yan, O. Ruwase, and R. Fonseca, "HyperDrive: Exploring hyperparameters with POP scheduling," in *Proc. 18th ACM/IFIP/USENIX Middlew. Conf.*, 2017, pp. 1–13.

[4] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA," in *Automated Machine Learning*. Cham, Switzerland: Springer, 2019, p. 81–95.

[5] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," 2016. [Online]. Available: arXiv:1603.06560.

[6] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*. Berlin, Germany: Springer, 2019.

[7] W. Xiao *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *Proc. 13th USENIX Symp. Oper. Syst. Design Implement. (SDI)*, 2018, pp. 595–610.

[8] W. Wang *et al.*, "Rafiki: Machine learning as an analytics service system," *Proc. VLDB Endow.*, vol. 12, no. 2, pp. 128–140, 2018.

[9] A. Saeed *et al.*, "Eiffel: Efficient and flexible software packet scheduling," in *Proc. USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2019, pp. 17–32.

[10] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Pias: Practical information-agnostic flow scheduling for commodity data centers," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 1954–1967, 2017.

[11] L. Jose, S. Ibanez, M. Alizadeh, and N. McKeown, "A distributed algorithm to calculate max-min fair rates without per-flow state," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 2, p. 21, 2019.

[12] Y. Guo, Z. Wang, H. Zhang, X. Yin, X. Shi, and J. Wu, "Joint optimization of tasks placement and routing to minimize coflow completion time," *J. Netw. Comput. Appl.*, vol. 135, pp. 47–61, Jun. 2019.

[13] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: Near-optimal network design for coflows," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 16–29.

[14] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2014, pp. 583–598.

[15] T. Chen *et al.*, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015. [Online]. Available: arXiv:1512.01274.

[16] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[17] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[18] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[19] K. G. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2016, pp. 240–248.

[20] M. Jeon, S. Venkataraman, J. Qian, A. Phanishayee, W. Xiao, and F. Yang, "Multi-tenant GPU clusters for deep learning workloads: Analysis and implications," Microsoft Research, Redmond, WA, USA, Rep., 2018. [Online]. Available: https://www.microsoft.com/en

[21] M. Blot, D. Picard, N. Thome, and M. Cord, "Distributed optimization for deep learning with gossip exchange," *Neurocomputing*, vol. 330, pp. 287–296, Feb. 2019.

[22] H. Zhang *et al.*, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2017, pp. 181–193.

[23] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, 2014, pp. 431–442.

[24] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *Proc. IEEE INFOCOM IEEE Conf. Comput. Commun.*, 2018, pp. 864–872.

[25] Z. Wang *et al.*, "Efficient scheduling of weighted coflows in data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 9, pp. 2003–2017, Sep. 2019.

[26] M. X. Goemans, "Improved approximation algorithms for scheduling with release dates," Assoc. Comput. Mach., New York, NY, USA, Rep., 1997.

[27] A. Singh *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 183–197, 2015.

[28] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[29] M. Queyranne, "Structure of a simple scheduling polyhedron," *Math. Program.*, vol. 58, nos. 1–3, pp. 263–285, 1993.

[30] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan, "Minimizing the sum of weighted completion times in a concurrent open shop," *Oper. Res. Lett.*, vol. 38, no. 5, pp. 390–395, 2010.

[31] S. Ahmadi, S. Khuller, M. Purohit, and S. Yang, "On scheduling coflows," in *Proc. Int. Conf. Integer Program. Combinatorial Optim.*, 2017, pp. 13–24.
[32] A. Agnetis, H. Kellerer, G. Nicosia, and A. Pacifici, "Parallel dedicated machines scheduling with chain precedence constraints," *Eur. J. Oper. Res.*, vol. 221, no. 2, pp. 296–305, 2012.
[33] H.-C. Hong and B. M. Lin, "Parallel dedicated machine scheduling with conflict graphs," *Comput. Ind. Eng.*, vol. 124, pp. 316–321, Oct. 2018.
[34] A. Agnetis, M. Flamini, G. Nicosia, and A. Pacifici, "Scheduling three chains on two parallel machines," *Eur. J. Oper. Res.*, vol. 202, no. 3, pp. 669–674, 2010.
[35] Y. Lu *et al.*, "One more queue is enough: Minimizing flow completion time with explicit priority notification," in *Proc. IEEE INFOCOM IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[36] K. H. Chan, J. Babiarz, and F. Baker, "Configuration guidelines for diffserv service classes," Internet Eng. Task Force, RFC 4594, 2006.
[37] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3366–3380, Nov. 2016.
[38] S. Luo, H. Yu, K. Li, and H. Xing, "Efficient file dissemination in data center networks with priority-based adaptive multicast," *IEEE J. Sel. Areas Commun.*, early access, Apr. 8, 2020, doi: 10.1109/JSAC.2020.2986616.
[39] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: Off-line and on-line approximation algorithms," *Math. Oper. Res.*, vol. 22, no. 3, pp. 513–544, 1997.
[40] N. Garg, A. Kumar, and V. Pandit, "Order scheduling models: Hardness and algorithms," in *Proc. Int. Conf. Found. Softw. Technol. Theor. Comput. Sci.*, 2007, pp. 96–107.
[41] S. Khuller, J. Li, P. Sturmfels, K. Sun, and P. Venkat, "Select and permute: An improved online framework for scheduling to minimize weighted completion time," *Theor. Comput. Sci.*, vol. 795, pp. 420–431, Nov. 2019.
[42] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," in *Proc. SIGCOMM Demonstration*, 2008, p. 527.
[43] J. Gu *et al.*, "Tiresias: A GPU cluster manager for distributed deep learning," in *Proc. 16th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2019, pp. 485–500.
[44] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. 13th EuroSys Conf.*, 2018, p. 1–14.
[45] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE INFOCOM IEEE Conf. Comput. Commun.*, 2019, pp. 505–513.
[46] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "SLAQ: Quality-driven scheduling for distributed machine learning," in *Proc. Symp. Cloud Comput.*, 2017, pp. 390–404.
[47] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 443–454, 2014.
[48] A. S. Schulz, "Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds," in *Proc. Int. Conf. Integer Program. Combinatorial Optim.*, 1996, pp. 301–315.

Pan Zhou is currently pursuing the Ph.D. degree in communication and information systems with the University of Electronic Science and Technology of China. His research areas include networks, cloud computing, distributed systems, and machine learning.

Hongfang Yu received the B.S. degree in electrical engineering from Xidian University in 1996, and the M.S. and Ph.D. degrees in communication and information engineering from the University of Electronic Science and Technology of China (UESTC), in 1999 and 2006, respectively. From 2009 to 2010, she was a Visiting Scholar with the Department of Computer Science and Engineering, University at Buffalo. She is currently a Professor with UESTC and Peng Cheng Laboratory. Her research interests include network survivability, network security, and next generation Internet.

Gang Sun (Member, IEEE) is an Associate Professor of computer science with the University of Electronic Science and Technology of China. His research interests include network virtualization, cloud computing, parallel and distributed systems, ubiquitous/pervasive computing and intelligence, and cyber security. He has edited special issues at top journals, such as *Future Generation Computing Systems* and *Multimedia Tools and Applications*. He has served as a Reviewer for the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE COMMUNICATIONS LETTERS, *Information Fusion*, *Future Generation Computing Systems*, the *Journal of Network and Computer Applications*, the *Journal of Supercomputing*, and the *Journal of Parallel and Distributed Computing*. He is a member of IEEE Computer Society.