Knowledge-Based Systems 201–202 (2020) 106002

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

# Online job scheduling for distributed machine learning in optical circuit switch networks

Ling Liu [a], Hongfang Yu [a,b,*], Gang Sun [a,**], Huaman Zhou [a], Zonghang Li [a], Shouxi Luo [c]

[a] Key Laboratory of Optical Fiber Sensing and Communications (Ministry of Education), University of Electronic Science and Technology of China, Chengdu, People's Republic of China
[b] Peng Cheng Laboratory, Shenzhen, People's Republic of China
[c] Southwest Jiaotong University, Chengdu, People's Republic of China

ARTICLE INFO

*Article history:*
Received 18 February 2020
Received in revised form 25 April 2020
Accepted 5 May 2020
Available online 12 May 2020

*Keywords:*
Distributed machine learning (DML)
Optical circuit switch (OCS)
Online job scheduling
Weighted Job Completion Time (WJCT)

ABSTRACT

Networking has become a well-known performance bottleneck for distributed machine learning (DML). Although lots of works have focused on accelerating the communication process of DML, they ignore the impact of the physical network on the DML performance. Concurrently, optical circuit switches (OCSes) are increasingly applied in data centers and clusters, which can fundamentally improve DML performance. It is worth noting that the non-negligible OCS reconfiguration delay makes OCS scheduling algorithms have a great impact on the upper application performance. However, existing OCS scheduling solutions are not suitable for DML jobs due to the iterative nature of DML jobs and their interleaving characteristics of communication and computation stages. Therefore, in this paper, we study the online multi-job scheduling for DML in OCS networks. Firstly, we propose heaviest-load-first (HLF), a heuristic algorithm for intra-job scheduling, which is based on the fact that the completion time of flows on the heaviest load port has a significant impact on the job completion time. Furthermore, we present Shortest Weighted Remaining Time First (SWRTF) algorithm for inter-job scheduling. In SWRTF, an *available* DML job is scheduled when the served job moves from communication stage to the computation stage, which significantly improves the circuit utilization. Based on large-scale simulations, we demonstrate HLF can significantly reduce the iteration communication time by up to 64.97% compared to the state-of-the-art circuit scheduler Sunflow. Besides, SWRTF can save up to 42.9%, 54.2%, 27.2% of Weighted-Job-Completion-Time (WJCT) compared to Shortest-Job-First, Baraat and Weighted-First inter-job scheduling algorithms, respectively.



## 1. Introduction

With the ever-increasing sizes of training data and models, distributed machine learning (DML) is becoming a prerequisite for solving large-scale machine learning problems [1]. In general, most DML training jobs work in an iterative fashion, and it may require hundreds of thousands or even millions of iterations to achieve ideal model accuracy [2]. In each iteration, local computation is firstly executed by every worker, and then model synchronization among all workers is followed. With the use of hardware accelerators, such as GPU [3], workers can deal with more training samples in unit time, leading to more frequent model synchronization. This puts forward a higher requirement for network throughput. The bottleneck of DML has shifted from computation to communication [4–6].

Although a large quantity of works have been proposed to accelerate DML training, such as resource scheduling [2,7] and parameter propagation optimization [8,9], they ignore the impact of the physical network on DML training, which has been analyzed in [10,11]. Concurrently, optical circuit switch (OCS) is increasingly deployed into data centers and clusters due to its high link rate, high reliability and low power consumption [12,13]. Besides, the OCS allows to reconfigure the network topology based on the traffic demand, leading to a *demand-aware* network [14].

The reconfigurable nature of OCS can greatly improve the network throughput and the application performance [15,16]. Apparently, the OCS-based network will also benefit DML training, which will be shown by an example in Section 2. Note that, two features of OCS should be considered, one is *port constraint* which means one input/output can only set up one circuit to one output/input port at a time. In other words, if multiple flows share the same port, they have to be scheduled one by one. The other is *reconfiguration delay*, the fixed time delay (milliseconds

* Correspondence to: No.2006, Xiyuan Ave, West Hi-Tech Zone, 611731, University of Electronic Science and Technology of China, Chengdu, Sichuan, People's Republic of China.
** Corresponding author.
*E-mail addresses:* yuhf@uestc.edu.cn (H. Yu), gangsun@uestc.edu.cn (G. Sun).

https://doi.org/10.1016/j.knosys.2020.106002
0950-7051/

or even tens of milliseconds [12,17]) to set up a new circuit. This means during the reconfiguration delay no traffic on the two ports can be transmitted. Therefore, the OCS scheduling algorithm dealing with port connections is particularly important because unreasonable scheduling may bring more reconfiguration delays [18,19], which will affect the job completion time (JCT).

However, existing OCS scheduling algorithms, such as coflow-based scheduling [18–20], are not applicable to DML. In general, most DML jobs require a large number of iterations to achieve ideal accuracy, and each iteration will generate a set of flows for model synchronization. In other words, there will be a series of these flows arriving and leaving within the lifespan of each DML job. Thus, DML job is a multi-stage job with multiple communication and computation stages. However, although existing coflow-based scheduling policies, such as shortest-coflow-first [18], can optimize the coflow completion time (CCT), they are not very well suitable for DML jobs. Because there is a large divergence between CCT and JCT for multi-stage job, such as DML training job, and minimizing CCT is not the same as minimizing JCT [21]. Therefore, existing OCS scheduling schemes cannot be used for DML training jobs directly.

Therefore, in order to reduce the DML training time, it is urgent to design a dedicated OCS scheduling scheme for DML. Considering the specificity of DML training jobs, we face the following difficulties.

- Single stage coflow scheduling in OCS networks has been proven to be NP-hard [18], while DML jobs contain lots of dependent communication stages, which makes the problem more difficult.
- The computation stage and communication stage of a DML job interlace each other. Even in traditional packet-switch networks, most multi-stage job scheduling schemes [21,22] only involve communication stages. How to effectively use the stagger of computation stage and communication stage to improve the circuit utilization so as to speed up training is the key to OCS scheduling algorithm for DML.

Accordingly, in this paper, we address the problem of online multi-job scheduling for DML in OCS networks and we propose a heuristic algorithm which consists intra-job scheduling and inter-job scheduling. Specifically, the major contributions are as follows:

- To our best knowledge, this is the first work that systematically studies how to schedule multi-DML-jobs in OCS networks, so that the total Weighted Job Computation Time (WJCT) can be minimized.
- For intra-job scheduling, we propose Heaviest-Load-First (HLF) which prioritizes the flows on the heaviest-load ports. HLF is based on this fact that the completion time of the flows on the heaviest ports in each iteration is more critical to the iteration communication time. For inter-job scheduling, we design an online method, Shortest Weighted Remaining Time First (SWRTF). In SWRTF, when a job which takes up the optical circuits completes its communication stage, the highest priority job will be scheduled according to HLF, which can effectively takes advantage of the interleaved communication and computation characteristics of DML jobs.
- We conduct large-scale simulations to evaluate the performance of HLF and SWRTF. The simulation results indicate that HLF can reduce up to 64.97% of the average iteration communication time (ICT) compared to Sunflow [18]. Compared with extensively used Shortest-Job-First (SJF) and Baraat [23], SWRTF saves WJCT by up to 42.9% and 54.2%, respectively. Besides, the algorithm combining HLF and SWRTF can also achieve very good performance.

**Table 1**
Job information.

| job ID | #worker | #parameter server | model size (Gb) |
|---|---|---|---|
| 1 | 5 | 1 | 5 |
| 2 | 5 | 1 | 6 |

The remainder of this paper is organized as follows. Section 2 motivates the background and the need of our work. Section 3 describes the details of the problem formulation. Section 4 presents our algorithms and analysis. Section 5 carries out extensive simulations to evaluate the performance of our algorithms. Section 6 describes the related works. Finally, Section 7 concludes this work.

## 2. Background and a motivating example

**Optical Circuit Switch (OCS)**. Compared with traditional packet switches, OCS can support higher link bandwidth (>100 Gbps) and consumes lower energy consumption (usually an order of magnitude less than packet switch [18]), because light is passively redirected from one port to another, independent of data rate [12]. For example, an electrical packet switch consumes 4 W/port while an OCS only consumes 150 mW/port. In addition, OCS has a large mean failure time and is more reliable, for example, up to 30 years for a commercial 3D-MEMS switch [18]. Besides, by changing the port connection states, a new topology can be obtained based on traffic demand, which makes better use of bandwidth and is conducive to the improvement of application performance.

With the rapid development of optical technology, an OCS will scale to thousands of input/output ports, for example, 1424 × 1424 [24], and the reconfiguration delay can be as low as microseconds or even nanoseconds [25–27]. Therefore, OCS can help to construct a large-scale network. In general, in OCS networks (also called OCS-based networks), all Top of Rack (ToR) switches can be connected to OCS [15–17,28], so they can communicate to each other directly by adjusting the connection states of OCS ports, without passing through the congested aggregation and core switches. For example, in Fig. 1(b), by setting up a circuit between ToR switch U1 and U3, they can transmit data directly through OCS, without going through the congested core link (U19↔U13) in fixed packet-switched network (Fig. 1(a)). After their data transmission, the circuit is torn down to provide for other data transmission.

Therefore, through dynamically changing topology, OCS can effectively alleviate the congestion within the network. Besides, many networks in industry introduce bandwidth oversubscription in order to lower the total cost [29,30]. When the oversubscription is larger, the network congestion is more likely to occur. In this case, the performance improvement brought by OCS will be more remarkable. Besides, the reconfigurable feature of OCS can also benefit different kinds of traffic through changing the network topology, therefore improving the application performance.

**A motivating example.** To illustrate the benefit of OCS networks for DML jobs, we give a motivating example of the performance of DML in traditional packet-switch networks (e.g., Fat-Tree) and OCS networks. The information of two jobs based on parameter server (PS) architecture [1] is shown in Table 1 and each job contains only 1 iteration and has the same weight 1. Two jobs start at time 0 and the computation time of two jobs is 1. For simplicity, we assume that the communication stage of PS architecture contains *push* phase and *pull* phase, regardless of the overlap of the two phases. Besides, other schemes reducing the data transmitted are not used, such as gradient compression

**Table 2**
WJCT comparision.

| Scheduling Solutions | Bandwidth (Gbps) | WJCT |
|---|---|---|
| OCS-based | 20 | 7.58 |
| OCS-based | 10 | 13.08 |
| SJF | 10 | 15.5 |
| SBP | 10 | 24 |

and quantization [31,32]. So the transmission data size of each worker is equal to model size.

The deployments of jobs in FatTree and OCS networks are shown in Fig. 1, where workers of job 1 and job 2 are deployed on servers under ToR switch U1/U5 and U2/U6, respectively. We assume servers have sufficient bandwidth, which can be achieved by adding multiple high-speed network cards to servers [4]. The paths from workers to parameter servers (PSs) of job 1 and 2 in traditional networks are shown in Fig. 1.

In FatTree, we exploit two scheduling schemes, Shortest Job First (SJF) and Share Bandwidth Proportionally (SBP) where two jobs can transmit data at the same time and share the bandwidth in proportion to their flow sizes. Note that, in OCS network, whether SJF or SBP is used, the job scheduling order is the same because there are no competing OCS ports between the two jobs. But different flows in each job will compete for the OCS ports connecting PS. Therefore, data transmission of two jobs need one circuit switching. More specifically, for job 1, first, OCS port 1 is connected to 3. When the data transmission from port 1 to 3 completes, the circuit between 1 and 3 is torn down and the circuit between 5 and 3 is established. Meanwhile, for job 2, OCS ports 6 and 2 also set up circuits with port 4 in turn.

When the reconfiguration delay of OCS is 0.01s, different scheduling solutions are shown in Fig. 2 where OCS-based is the job scheduling algorithm in OCS network. The WJCT is shown in Table 2. It is observed that even when each ToR switch is only connected to one OCS port, bandwidth oversubscription of FatTree is 1, and the bandwidth of packet-switch and OCS is 10 Gbps, the WJCT of OCS-based solution is $\frac{15.5-13.08}{15} \times 100\% \approx 15.61\%$ and $\frac{24-13.08}{24} \times 100\% \approx 45.5\%$ less than that of SJF and SBE, respectively. And when the circuit bandwidth is 20 Gbps, the values increase to 51.1% and 68.4%. It is worth noting that when one ToR switch is connected to multiple OCS ports, U1/U2 and U5/U6 can communicate with U3/U4 at the same time. Hence, when bandwidth is 10 Gbps, the JCTs of job 1 and 2 are 4.01, 4.61, respectively. And the WJCT is $\frac{15.5-8.62}{15.5} \times 100\% \approx 44.4\%$ less than that of SJF. With the development of fast optical switch, reconfiguration delay can be as low as microsecond or nanosecond [25–27]. Actually the OCS bandwidth can be more than 100 Gbps [13]. The larger bandwidth and lower reconfiguration delay are more conducive to improve the training performance of DML.

Therefore, the dynamic reconfigurable nature of OCS and high bandwidth can greatly improve the DML training performance. However, *port constraint* and reconfiguration delay of OCS [19] enable the OCS scheduling algorithms to create different reconfiguration delays and job completion times. Therefore, designing reasonable OCS scheduling algorithms are vitally important for DML training performance improvement.

## 3. Problem formulation

### 3.1. Network model

Similar to [18], the entire network fabric is abstracted as one non-blocking N×N-port OCS with link bandwidth *B*. This model is simple but practical because topologies used in industry, such as Fat-tree or Clos [29,30], which enables to build a network

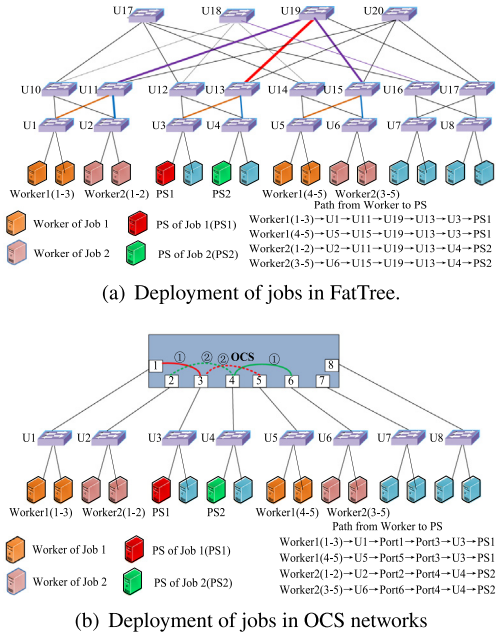


**Fig. 1.** Deployments of jobs in FatTree and OCS networks. (a) Three workers Worker1 (1–3) and two workers Worker1 (4–5) of Job 1 are deployed on servers under ToR U1 and U5, respectively. Two workers Worker2 (1–2) and three workers Worker2 (3–5) of Job 2 are deployed on servers under ToR U2 and U6, respectively. So the data sizes transmitted from U1/U5 to PS1 and U2/U6 to PS2 are 15/10 and 12/18, respectively. (b) only ToR switches connected to OCS are shown and other packet-switched network is not shown. Each ToR switch connects to an OCS port and ToR switches can be connected to each other directly by adjusting the connection states of OCS port. The circuits are set up as follows: first, port 1/6 is connected to 3/4 (①, the red/green solid line), then port 5/2 is connected to 3/4 (②, the red/green dotted line). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

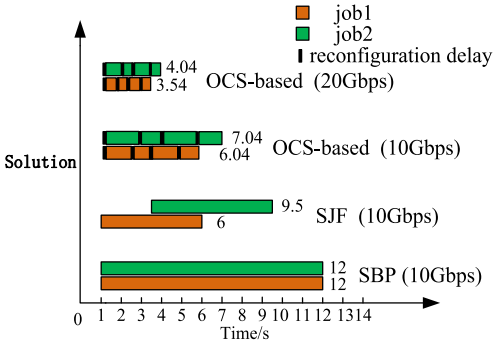


**Fig. 2.** Results of different scheduling solutions.

with full bisection bandwidth. The switch ports are connected to ToR switches, and each ToR switch is connected to one or several ports of the optical switch through optical multiplexers. Besides, each ToR switch is connected to a group of servers. Note that, in our network model, we assume the bandwidth of servers will not be a performance bottleneck. This is reasonable because we can exploit the same approach as in [4], which is to add multiple network interfaces to the servers. It is worth noting that the circuit network is suitable for "elephant" flows due to its nonnegligible reconfiguration delay. Since machine learning training model is often large enough (e.g. VGG-16 is 528 MB), the circuit network naturally fits DML applications. In practice, our network infrastructure is always a hybrid style consisting of both circuit switch and packet switch. The packet-switched network is

mainly responsible for the transmission of "mice" flows, such as DML control messages. Note that, the packet-switched network does not affect our scheduling algorithm, so we just have to focus on circuit-switched network.

There are two optical switch models, namely *all-stop* model where all circuits should be torn down during circuit reconfiguration, and *not-all-stop* model where unchanged circuits are not impacted and can keep data transmitting during circuit reconfiguration. In this work, we adopt the *not-all-stop* model due to its better circuit utilization with fewer preemptions [18] . As in [18,19], we also assume all the data buffering occurs on the edge of the network, such as ToR switches and sender machines, due to no buffering in OCS.

### 3.2. Traffic model

Parameter Server (PS) [11,33] and AllReduce [34,35] are two extensively used architectures for model synchronization. In PS architecture, servers are divided into workers which are responsible for local training and parameter servers (PSs) which are responsible for model synchronization. In MPI AllReduce architecture, such as Ring [36] and Butterfly [37], there is no central node responsible for parameter aggregation. Instead, all nodes are equal and model parameters can be synchronized with each other through some rules.

In this paper, we do not restrict which communication architecture DML is executed on, but we assume the synchronous mode is exploited, which is widely used to enable reproducibility in ML cluster [38] because of its guarantee for the model convergence. In synchronous mode, in AllReduce architecture, only when all workers receive the latest model parameters will the computation stage of the next iteration begin. Similarly, each worker in PS architecture computes and pushes parameter updates (i.e., gradients) to PSs (*push* stage), and it will not begin the next iteration until it receives the latest model from PSs (*pull* stage).

For DML training jobs, each iteration contains a communication phase and a computation phase, and we define all flows generated during an iteration as *iterationflow*. It is worth noting that the communication of the current iteration overlaps partially with the computation of the next iteration [6]. But that does not affect our definition, because it is only the equivalent of less computation time and the computation time can be regarded as fixed during training. Besides, some flows in *iterationflow* must be completed earlier. So an *iterationflow* can be divided into several stages according to the parameter propagation scheme where parameters of some layers may have high transmission priority [8]. For example, most *push* flows in PS architecture must be scheduled before *pull* flows, except for the flows that overlap each other [6], thus the stages of these *push* flows can be set to 1, the overlapped flows to 2, and other *pull* flows to 3. The stages can be obtained according the parameter propagation scheme [6,8]. Similarly, the stages of flows in AllReduce architecture [36] can also be gained because the communicating servers and data sizes transmitted in each step are known.

In synchronization mode, a job finishes after reaching its iteration number or achieving ideal model accuracy. In our traffic model, we assume each DML job is set an iteration number. Since our network is abstracted as one non-blocking N× N-port OCS, the traffic demand of a DML task is expressed by a two-tuples <*D*, *IterNum*>, where $D$ is the demand matrix and *IterNum* indicates iteration number. Besides, each element of $D$ is a two-tuples $<d_{ij}, k_{ij}>$, where $d_{ij}$ indicates flow $f_{ij}$ should transfer $d_{ij}$ amount of data from input port $i$ to output port $j$ in each iteration, $k_{ij}$ indicates the scheduling stage of the flow $f_{ij}$ and the smaller $k_{ij}$ represents the flow should be scheduled earlier. Note that in this paper, the flow represents the data from one OCS port to another one, so the data of each flow may be generated by a different number of workers, which depends on the deployments of workers or PSs in the network. Therefore, the flow sizes are not the same and can vary greatly, which is different from the assumptions in [18].

Given the deployments of workers or PSs, the OCS input/output ports of the paths can be determined. But since there may be competing ports between different flows, and an OCS output port can only connect to one input port at the same time, so the OCS scheduling algorithm should decide when to start or stop a flow within and between DML jobs.

### 3.3. DML jobs scheduling model

Usually in production systems, more important jobs are prioritized by setting a larger weight value. Accordingly, we consider that a set of jobs $\{1, 2, \ldots, N\}$with weight $\{W_1, W_2, \ldots, W_N\}$ are submitted to the OCS networks. Each job contains multiple iterations, and each iteration consists of computation phase and communication phase. In each iteration, a set of flows (*iterationflow*) are needed to transmitted through OCS. In general, OCS scheduling algorithms are divided into preemptive and non-preemptive approaches. Preemptive solutions require extra reconfiguration overhead to re-establish circuits to serve a flow [18]. Therefore, we only consider non-preemptive scheduling algorithms in the sense that when two flows share the same port, only when one flow completes the transfer can the other be started.

Our objective is to minimize WJCT. Although we focus on the online DML job scheduling in OCS networks, we first present its offline version as follows. The notations are shown in Table 3.

$$\min \quad \sum_{n=1}^{N} W_n J_n \tag{1}$$

s.t.

$$J_n = z_n^{K_n}, \tag{2}$$

$$z_n^i = \max f_{nr}^i, \forall r \in \Gamma_n^i, \forall i \in [K_n], \forall n \in [N], \tag{3}$$

$$h_n^i = \min g_{nr}^i, \forall r \in \Gamma_n^i, \forall i \in [K_n], \forall n \in [N], \tag{4}$$

$$z_n^i + T_n \le h_n^{i+1}, \forall i \in [K_n - 1], \forall n \in [N], \tag{5}$$

$$f_{nr}^i = g_{nr}^i + \delta + \frac{S_{nr}}{B}, \quad \forall r \in \Gamma_n^i, \forall i \in [K_n], \forall n \in [N], \tag{6}$$

$$f_{nr'}^i \le g_{nr''}^i, \forall r', r'' \in \Gamma_n^i, r' \prec r'', \tag{7}$$

$$\sum_{r' \in \Gamma_{n'}^j} \sum_{d \in \Gamma_n^i, d \ne r'} \sum_{n' \in [N]} I(d \eth r')(y_{nd}^i(t) + y_{n'r'}^j(t)) = 1,$$

$$g_{nd}^i \le t \le f_{nd}^i, \forall j \in [K_{n'}], \forall i \in [K_n], \forall n \in [N], \tag{8}$$

$$y_{nr}^i(t) = \begin{cases} 1, & f_{nr}^i \le t \le g_{nr}^i \\ 0, & others \end{cases}, \forall i \in [K_n], \forall r \in \Gamma_n^i, \forall n \in [N], \tag{9}$$

$$g_{nr}^0 \ge T_n + A_n, \forall r \in \Gamma_n^0, \forall n \in [N], \tag{10}$$

$$z_n^i, h_n^i, g_{nr}^i, f_{nr}^i \ge 0, \forall r \in \Gamma_n^i, \forall i \in [K_n], \forall n \in [N], \tag{11}$$

As shown in Eq. (2), the job completion time is equal to the completion time of the last iteration. (3) and (4) respectively give the completion time and start time of an *iterationflow*, which are equal to the time of the latest flow to finish and the time of the earliest flow to start, respectively. Constraint (5) ensures the start time of the (i+1)*th iterationflow* is greater than or equal to

**Table 3**
Notation.

| Symbol | Definition |
|---|---|
| $N$ | # of jobs |
| $K_n$ | # of iteration of job $n$ |
| $A_n$ | start/arrive time of job $n$ |
| $\Gamma_n^i$ | flow set of job $n$ in its $i$*th* iteration |
| $S_{nr}$ | size of flow $r$ of job $n$ |
| $W_n$ | weight of job $n$ |
| $B$ | bandwidth of circuit |
| $\delta$ | reconfiguration delay |
| $T_n$ | computation time of job $n$ |
| $y_{nr}^i(t)$ | whether or not flow $r$ of job $n$ in the $i$*th* iteration is transmitting at time $t$ |
| $J_n$ | completion time of job $n$ |
| $z_n^i$ | completion time of $i$*th iterationflow* of job $n$ |
| $h_n^i$ | start time of $i$*th iterationflow* of job $n$ |
| $f_{nr}^i$ | completion time of flow $r$ of $i$*th iterationflow* of job $n$ |
| $g_{nr}^i$ | start time of flow $r$ of $i$*th iterationflow* of job $n$ |

the completion time of the $i$*th iterationflow* plus the computation time of the job. (6) indicates the completion time of a flow equals to the start time of the flow plus the transfer time and the switching delay, which indicates that every flow only needs one circuit switching and preemption is not allowed. Constraint (7) upper bounds the completion time of flows with smaller stage by the start time of flows with bigger stage in each iteration of a DML job, where $r' \prec r''$ means the stage of flow $r'$ is smaller than that of $r''$ and $r'$ should be scheduled before $r''$. (8) guarantees that other flows sharing a port with a flow cannot be transferred at the same time if the flow is transmitting, where the indicator $I(d\eth r')$ denotes whether flow $d$ and $r'$ share an OCS input or output port. (9) presents $y_{nr}^i(t)$ is set to 1 only when the flow is transmitting. (10) denotes the flow of the first *iterationflow* starts only when job has completed computation stage.

**Theorem 1.** *Problem* (1) *is* NP-Hard.

**Proof.** Considering a special case of problem (1) when there is only one DML job in the network. When the job has only one iteration and the stage number of *iterationflow* is 1, then problem (1) becomes the non-preemptive circuit scheduling for one coflow [18], which can reduce to the *NP-hard* problem, non-preemptive open-shop problem to minimize work span. In fact, our problem consists of multiple jobs, each of which contains sequential *iterationflow* communication stages and computation stages, which is much more difficult to schedule than a single *iterationflow*. Consequently, the original problem (1) is also *NP-hard*.

## 4. Algorithm design and analysis

Since solving the off-line multi-job scheduling in OCS networks is mathematically intractable, it is much more difficult to solve the online version. So we design a heuristic algorithm, which contains intra-job scheduling and inter-job scheduling. As any other job scheduling algorithms in packet-switched network [21,39] or circuit-switched network [18,19,40], only a job is scheduled in OCS networks at every moment. It is worth noting that we also pursue work-conserving property by distributing the idle circuits to other jobs, which will be described later.

### 4.1. Intra-job scheduling algorithm

In fact, the intra-job scheduling problem is the *iterationflow* scheduling problem, and our objective is to minimize the *iterationflow* completion time or iteration communication time (ICT). Given a non-negative matrix $D = [\langle d_{ij}, k_{ij}\rangle] \in R^{M\times M}$, where $M$ is the port number of the OCS, our scheduler needs to decide *when* to serve *which* flow until all flows are completed with consideration of the dependency of the flows.

**Algorithm 1:** HLF: Heaviest-Load-First

```
Input: Iterationflow Flows, stage number K, reconfiguration delay δ,
       bandwidth B
Output: circuit scheduling solution for each flow Flows, the iterationflow
        completion time T
Initialization:
// the CurrentTime is the time when OCS ports needs to be
   configured for flows
CurrentTime ← δ
IsPortAvailable ← 0
k ← 1
for b ∈ PortBusy do
    b.AvailableTime ← 0
end

while k ≤ K do
    StageFlows ← get flows with stage k from Flows
    while StageFlows ≠ ∅ do
        for b ∈ PortBusy do
            if b.AvailableTime ≤ CurrentTime then
                IsPortAvailable ← 1
            end
        end
        if IsPortAvailable == 1 then
            // find the flow to be scheduled
            (SelectedFlow, StartTime, EndTime) ←
            FindHeaviestPortFlow(StageFlows, PortBusy, CurrentTime, B, δ)
            flow ← find SelectedFlow ∈ Flows
            update the AvailableTime of input and output ports of flow
            ∈ PortBusy
            // update start/end time of the SelectedFlow
            flow.StartTime ← StartTime
            flow.EndTime ← EndTime
            IsPortAvailable ← 0
            remove flow from StageFlows
        end
        else
            // there are no available ports at this moment
            CurrentTime ← the earliest available time of all ports +δ
        end
    end
    k ← k + 1
end
T ← the largest available time of b ∈ PortBusy
return Flows,T
```

In the circuit-switched network, each flow incurs one circuit reconfiguration delay with non-preemptive solution. Thus the minimum data transfer time required for a flow $f_{ij}$ is

$$t_{ij} = \begin{cases} \dfrac{d_{ij}}{B} + \delta, & d_{ij} \neq 0. \\ 0, & d_{ij} = 0. \end{cases} \tag{12}$$

**Algorithm 2:** FindHeaviestPortFlow

**Input**: *StageFlows*, *PortBusy*, *CurrentTime*, *B*, $\delta$
**Output**: *SelectedFlow*, *StartTime*, *EndTime*

1 *StartTime* ← 0
2 *EndTime* ← 0
3 *HeaviestLoad* ← 0
4 *HeaviestPort* ← 0
// find the heaviest port
5 **for** $b \in$ *PortBusy* **do**
6  **if** *b.AvailableTime* ≤ *CurrentTime* **then**
7  compute the load of *b* according to *StageFlows*
8  **if** *HeaviestLoad* ≤ *b.Load* **then**
9  *HeaviestPort* ← *b*
10  *HeaviestLoad* ← *b.Load*
11  **end**
12  **end**
13 **end**
14 **if** *HeaviestPort is input port of flows* **then**
15  *HeaviestPort*1 ← find the corresponding available heaviest output port
16  *SelectedFlow* ← flow(*HeaviestPort* → *HeaviestPort*1)
17 **end**
18 **else**
19  *HeaviestPort*1 ← find the corresponding available heaviest input port
20  *SelectedFlow* ← flow(*HeaviestPort*1 → *HeaviestPort*)
21 **end**
22 *volume* ← volume of SelectedFlow
// update the available time of two ports
23 *StartTime* ← *max*(*PortBusy*[*HeaviestPort*].*AvailableTime*,
24 *PortBusy*[*HeaviestPort*1].*AvailableTime*)
25 *EndTime* ← $StartTime + \frac{volume}{B} + \delta$
26 *PortBusy*[*HeaviestPort*].*AvailableTime* and *PortBusy*[*HeaviestPort*1].*AvailableTime* ← *EndTime*
27 return *SelectedFlow*,*StartTime*,*EndTime*

The theoretical lower bound of ICT is

$$T_L = \sum_{k=1}^{K} I(k_{ij} = k) max(max_i \sum_{j=1}^{M} t_{ij}, max_j \sum_{i=1}^{M} t_{ij}). \quad (13)$$

where $K$ is the stage number of the *iterationflow* and the indicator $I(k_{ij} = k)$ denotes whether flow $f_{ij}$ belongs to stage $k$.

Apparently, $T_L$ depends on the OCS port with the heaviest load, where load of a port represents the sum of traffic on the port. Inspired by this, we present the Heaviest-Load-First (HLF) heuristic algorithm for intra-job scheduling. The basic process of HLF is as follows. The flows are scheduled in turn according to their stages, and the flows with smaller stage are scheduled earlier. In general, there are multiple flows in the same stage. In each stage, firstly, we compute the load of each port, and select the heaviest loaded port of all *available* ports $S$, where *available* ports mean the ones that are not occupied by other flows and can transfer data at this moment. If $S$ is the input/ output port of the flows, then the most heavily loaded output/input port $D$ is selected from the input/output ports through which these flows pass. Then, HLF makes reservation for the selected flow ($f_{SD}$/$f_{DS}$) and updates the remaining *iterationflow* demand. Next, HLF continues to schedule circuits for the remaining flows in this stage until all flows are completed. When all flows in this stage are completed, the flow scheduling of the next stage begins.

It is worth noticing that if we schedule light loaded port first, the flows on the heaviest loaded port still need to be transferred one by one, which will prolong the ICT. On the contrary, if flows on the heaviest loaded port are scheduled first, the other idle circuits can be exploited to transfer other flows, which is conducive to speed up the transfer of the *iterationflow*. Therefore, the heaviest loaded port has great impact on ICT.

In HLF, in order to minimize the reconfiguration delay for each circuit, once a flow starts to transmit data, it cannot be interrupted by other flows. Besides, the *not-all-stop* model of OCS makes each circuit can be set up or torn down without considering other circuits.

**Table 4**
Iterationflow information.

| | out.6 | out.7 | out.8 |
|---|---|---|---|
| in.1 | 25 | 12 | 0 |
| in.2 | 0 | 20 | 15 |
| in.3 | 0 | 25 | 20 |
| in.4 | 12 | 0 | 0 |

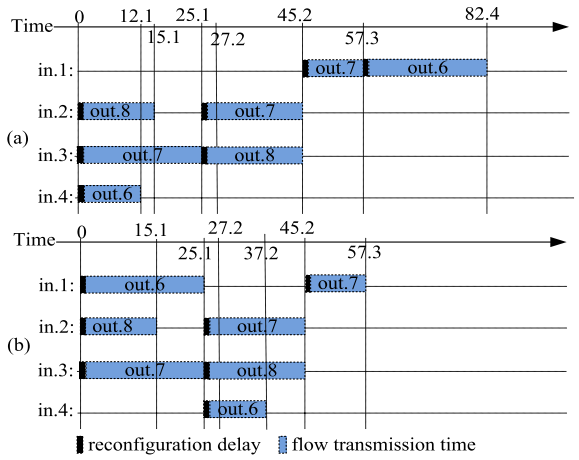


**Fig. 3.** Two intra-job scheduling algorithms. (a) Sunflow. (b) HLF. Suppose the bandwidth of circuit is 1, so the circuit duration is equal to the flow size.

Algorithm 1 and 2 show the pseudocode of HLF. HLF leverages a *PortBusy* data structure to record the latest available time (*AvailableTime*) of each port which indicates the time when the port can be used by other flows. When a flow is selected, the *AvailableTime* of the corresponding output port and input port should be updated (Line 20, HLF). At the same time, the start and end transfer times of the flow are recorded in the *Flows* data structure (Line 21–22, HLF). When all demand is drained, the maximum *AvailableTime* of all ports is the ICT (Line 32, HLF).

We use an easy example that compare HLF and Sunflow [18] to demonstrate the performance of HLF. There is only a stage in a given *iterationflow* shown in Table 4, so the *iterationflow* can be regarded as a *coflow*. Sunflow [18] is the first integrating coflow and circuit scheduling when the network is abstracted as a non-blocking *not-all-stop* OCS. The key idea of Sunflow is that flows are scheduled in a random order and preemption is also not allowed. Fig. 3 illustrates the key difference between HLF and Sunflow [18]. As shown in Fig. 3(b), according to HLF, the first scheduled flow is 3 → 7, and then 1 → 6, 2 → 8, etc. However, considering the arbitrary order of the selected flow in Sunflow, when the first selected flow is 2 → 8, and then 4 → 6, 3 → 7, etc. The ICTs of HLF and Sunflow are 57.3 and 82.4, respectively. And the theoretical lower bound of ICT is also 57.3. Obviously, HLF can significantly reduce ICT.

### 4.2. Inter-job scheduling algorithm

For inter-job scheduling, when multiple jobs compete for optical circuit resources, the system scheduler should decide *when* to serve *which* job. In this paper, we assume that all jobs work in synchronous mode and finish after a given iteration number. Some jobs will complete and some new jobs may be added over time in the network. Therefore, an online scheduling algorithm is desired.

Considering the interleaving of DML communication phase and computation phase, and the dynamic arrival of DML jobs, we propose an online inter-job scheduling algorithm Shortest

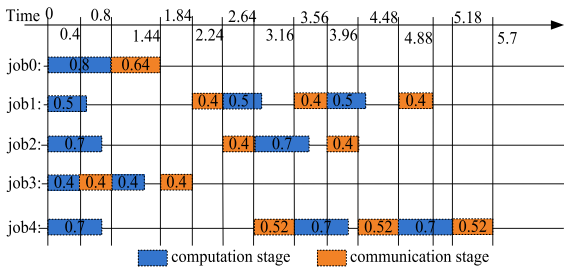


**Fig. 4.** Inter-job scheduling with SWRTF.

Weighted Remaining Time First (SWRTF). The key idea of SWRTF is that when the circuits are idle, we preferentially schedule the highest priority job among all available jobs. The priority of a job is calculated based on the Remaining Completion Time ($RCT_n$) and weight ($W_n$) of the job. Since each iteration time can be seen as the same and the remaining iteration number of a DML job is known, it is easy to get $RCT_n$. More specifically, the priority of a job $n$ is

$$P_n = \frac{W_n}{RCT_n}. \tag{14}$$

SWRTF is inspired by the shortest weighted processing time (SWPT) algorithm [41] where jobs are scheduled in non-decreasing order of the ratio of processing time to weight. The biggest difference between the two algorithms is that we consider DML jobs to be multi-stage, and each job needs to be scheduled multiple times to complete, instead of SWPT where a job only needs to be scheduled once.

Note that when a new job arrives, its iteration communication time (ICT) should be firstly computed according to HLF, and its priority can also be calculated. When the current job finishes communication stage and then enters computation stage, releasing all circuits, another available highest priority job can be scheduled. Therefore, SWRTF can more effectively leverage the spare time of circuits to transfer data of other jobs when the current job is in the computation phase, which greatly improves the circuit utilization. It is noteworthy that once a job has finished its current data transmission, its priority should be re-calculated. Algorithm 3 shows the pseudocode of SWRTF, where available time is the time when the job has completed the computation phase and is waiting for communication phase.

Fig. 4 shows an easy example of the inter-job scheduling process of SWRTF. The job information is illustrated in Table 5, and we assume the ICT is calculated by HLF. The WJCT of SWRTF is 44.6. If the weight of a job is taken as its priority, then the WJCT is 55.42. If we exploit the shortest-job-first (SJF) solution, the WJCT is 74.14. By contrast, our SWRTF performs the best.

### 4.3. Work conservation and starvation avoiding

In the previous sections, optical circuits are only allocated to the *iterationflow* of a DML job at any moment. However, there may be some idle circuits that can be used to serve more flows. To improve the circuit utilization, other flows of available DML jobs are scheduled on the idle circuits at the same time. Note that these flows cannot block the mainly served job, they should be interrupted to release circuits to the mainly served job. Therefore, work-conserving property is achieved. Naturally, we also use priority defined in SWRTF to select the available jobs to preempt the idle circuits, which is not shown in Algorithm 1.

Scheduling jobs based on priority may lead to starvation when priorities of some DML jobs are always smaller. In order to avoid starvation, we set time parameter $\tau$, and check whether there are any jobs that have not been scheduled during the last $\tau$ over time. If there are jobs that have not been served, these jobs should be scheduled first.

**Algorithm 3:** SWRTF: Shortest Weighted Remaining Time First

```
   Input: jobs J, starvation time τ
   Output: weighted job completion time T
1  Initialization:
2  CurrentTime ← the earliest available time of all jobs
   // the remaining jobs
3  RemJobs ← J
4  T ← 0
5  while RemJobs ≠ ∅ do
6      AvailableJobs ← find jobs whose available time ≤ CurrentTime
7      if AvailableJobs ≠ ∅ then
8          StarveJobs ← find jobs from AvailableJobs whose waiting time ≥ τ
9          StarveFlag ← 0
10         while StarveJobs ≠ ∅ do
11             StarveFlag ← 1
12             job ← find the highest priority the job from StarveJobs
13             execute Line 18-26
14             remove job from StarveJobs
15         end
16         if StarveFlag == 0 then
17             job ← find the highest priority job from AvailableJobs
18             CurrentTime ← CurrentTime + job.CommunicationTime
               // update the remain iteration number of the job in J
19             job.RemainIteration ← job.RemainIteration − 1
20             if job.RemainileIteration == 0 then
                   // the job has finished,update the end time of
                      the job in J
21                 job.EndTime ← CurrentTime
22                 remove job from remjobs
23             end
24             else
                   // update the available time of the job in J
25                 job.AvailableTime ← CurrentTime + job.ComputeTime
26             end
27         end
28         else
29             goto Line 5
30         end
31     end
32     else
33         EarliestJob ← find the earliest available job from RemJobs
34         CurrentTime ← EarliestJob.AvailableTime
35     end
36 end
37 for job ∈ J do
38     T ← T + job.Weight × job.EndTime
39 end
40 return T
```

### 4.4. Complexity analysis

HLF has a computational complexity of $O(MR_i)$, where $R_i$ is the flow number of *iterationflow* of a DML job $i$, and $M$ is the port number of OCS. Furthermore, the algorithm of finding the highest priority available job has a computational complexity of $O(N)$, where $N$ is the job number. Thus, SWRTF has a computational complexity of $O(\sum_{i=1}^{N} MR_i + \sum_{i=1}^{N} K_i N)$, where $K_i$ is the iteration number of job $i$.

## 5. Performance evaluation

To evaluate the performance of HLF and SWRTF, we implement a flow-level discrete-event simulator with various intra-job and inter-job scheduling algorithms. In our simulations, we consider a OCS with $M = 128$ input/output ports, and we use synthetic models to generate DML job workloads, such as weight, computation time and *iterationflow*. Besides, we assume that gradient compression [31], coding [42] and other methods aiming

**Table 5**
Job information.

| Job ID | Iteration number | Computation time | Communication time | Weight |
|---|---|---|---|---|
| 0 | 1 | 0.8 | 0.64 | 1 |
| 1 | 3 | 0.5 | 0.4 | 3 |
| 2 | 2 | 0.7 | 0.4 | 2 |
| 3 | 2 | 0.4 | 0.4 | 5 |
| 4 | 3 | 0.7 | 0.52 | 2 |

at reducing traffic are not used during DML training. Hence, the *iterationflow* for all iterations of a job can be regarded as the same.

### 5.1. Comparison of intra-job scheduling

In order to evaluate the performance of our intra-job scheduling algorithm, we compare HLF with the theoretical lower bound $T_L$ calculated by Eq. (11) and Sunflow [18] which has the same network model and the optical switch model. Note that we do not compare our algorithms with other algorithms using *all-stop* model, for example, Solstice, due to its worse performance than Sunflow [18]. More specifically, we compare the performance of HLF and the intra-coflow circuit scheduling algorithm of Sunflow (intra-Sunflow) in terms of *iterationflow* of DML model, bandwidth, reconfiguration delay and *iterationflow density* $r$. $r$ is defined as the average proportion of non-zero elements in the traffic demand matrix of all stages (e.g., for an *iterationflow* with 2 stages, the *iterationflow density* is 0.073 if the number of flows in stage 1 and 2 are 1000, 1200, respectively, i.e. $r = \frac{1}{2} \times (\frac{1000}{128\times128} + \frac{1200}{128\times128}) \approx 0.073$). Note that, we use Sunflow in each stage in order to compare it with HLF.

In this part, we mainly employ a typical deep learning model VGG-16 (528 MB) except the experiment to verify the impact of different *iterationflows* on the ICT where four DML models are used. Besides, except for experiment that tests the effect of different bandwidths on ICT, the circuit bandwidth in other experiments are all set to 100 Gbps. In experiment that analyzes the impact of circuit reconfiguration delay on ICT, the reconfiguration delay is set to 0.0001–0.1 s, and in other experiments, it is 0.01 s.

**Impact of various *iterationflows***: We firstly test the performance of HLF and intra-Sunflow in terms of *iterationflows* of four DML models when reconfiguration delay is 0.01 s. Considering the randomness of intra-Sunflow, we repeat the intra-Sunflow algorithm 20 times for each *iterationflow*. As shown in Fig. 5, each purple point represents the result of each test of intra-Sunflow. From this experiment, we can see that the ICT of HLF is 0.87% to 17.06% larger than $T_L$. In addition, it is apparent that HLF can achieve better performance for different *iterationflows* compared with intra-Sunflow. Specifically, HLF can save 20.29% to 64.97% of the average ICT compared to intra-Sunflow. This result indicates that priority scheduling of flows on ports with heavy loads can help improve the circuit utilization and reduce ICT.

**Impact of circuit bandwidth**: Fig. 6 shows the impact of optical circuit bandwidth on the performance of HLF and intra-Sunflow when reconfiguration delay is 0.01 s. Note that the performance of intra-Sunflow is averaged by 20 tries. Apparently, all ICTs show a downward trend with the increase of bandwidth, which indicates the bandwidth has a greater impact on ICT than flow scheduling order. Besides, HLF can achieve 41.08% lower ICT than intra-Sunflow on average and is just 13.22% larger than $T_L$. It is worth noting that the gap between intra-Sunflow, HLF, and $T_L$ decreases gradually with the increase of bandwidth. That is because when the reconfiguration delay is fixed, the delay time caused by switching circuits accounts for a smaller proportion of the total communication time (ICT) as bandwidth increases.

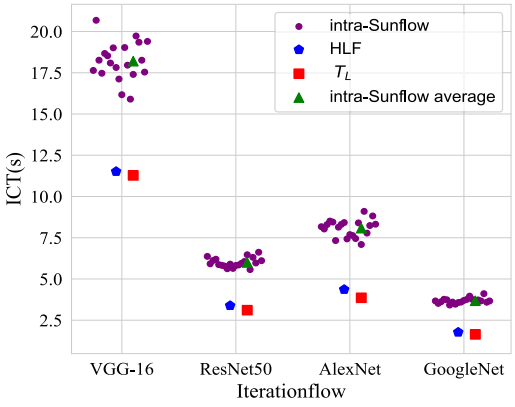


**Fig. 5.** ICT Comparison with various *iterationflows*.

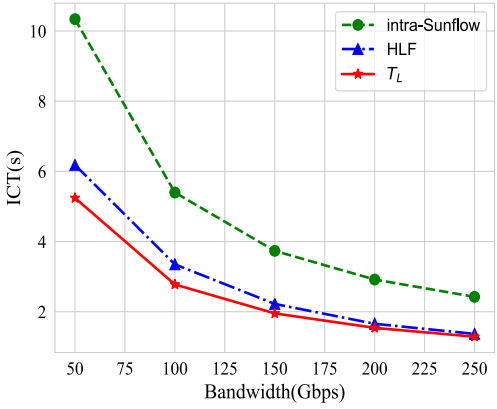


**Fig. 6.** ICT Comparison with different circuit bandwidths.

**Impact of reconfiguration delay**: Fig. 7 reflects how the reconfiguration delay affects ICT. For both algorithms, the larger reconfiguration delay leads to a longer ICT. As can be seen from the slope of intra-Sunflow and HLF in this figure, intra-Sunflow is much more sensitive to reconfiguration delay due to more circuit switching than HLF. For instance, when reconfiguration delay changes from 0.01 s to 0.1 s, the ICT obtained by HLF increases by 64.26%, while that of intra-Sunflow grows by 97.23%. In other words, HLF is more stable under different reconfiguration delays.

**Impact of *iterationflow density***: Fig. 8 indicates that as *iterationflow density* $r$ increases, the ICTs of all algorithms increase. Nonetheless, ICT of intra-Sunflow grows higher than that of HLF as density increases. For example, when $r$ changes from 0.192 to 0.497, ICT of intra-Sunflow goes from 25.93 s to 58.21 s and ICT increases by about 124%, while ICT of HLF changes from 14.92 s to 21.84 s and ICT only increases by about 46.4%. That is because larger *iterationflow density* means more flows and more intense port competition among flows, and scheduling flows randomly can lead to more circuit switching and therefore

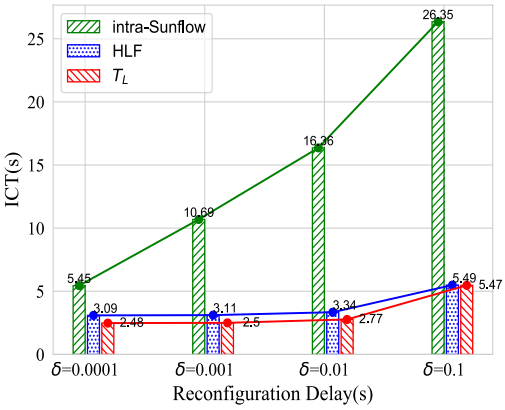


**Fig. 7.** ICT Comparison with different reconfiguration delays.

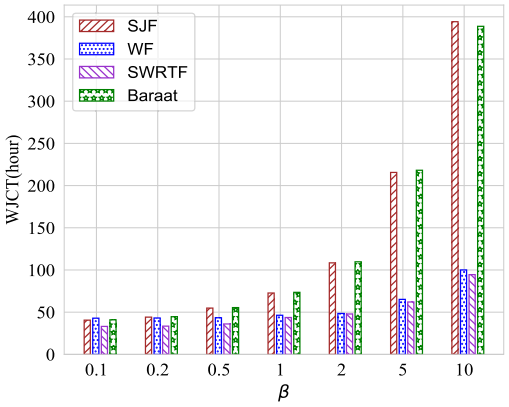


**Fig. 9.** WJCT Comparison with different ratios of the computation time to communication time $\beta$.

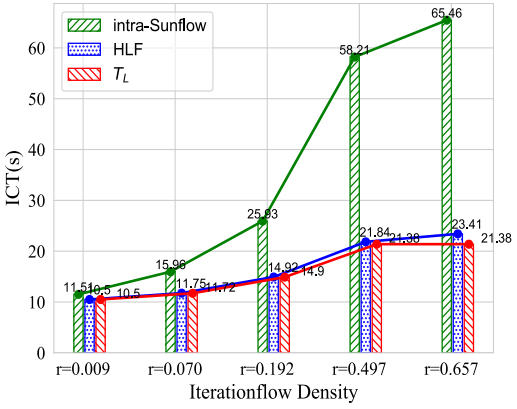


**Fig. 8.** ICT Comparison with different *iterationflow densities*.

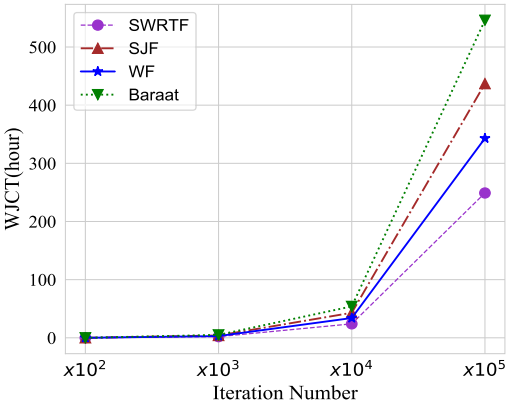


**Fig. 10.** WJCT Comparison with different iteration numbers.

longer ICT. Therefore, HLF also has a more stable performance for *iterationflows* with different *iterationflow densities*.

### 5.2. Comparison of inter-job scheduling

In this section, we compare SWRTF with three inter-job scheduling schemes, the first is the widely used algorithm Shortest-Job-First (SJF), which is also used for inter-coflow in Sunflow [18], the second is Baraat [23] which schedules different jobs in a FIFO manner, and the third is Weighted-First (WF) scheduling algorithm in which the weight of a job is taken as its priority directly. We compare these solutions in terms of $\beta$ (the ratio of computation time to communication time), iteration number and inter-job arrival interval.

In this part, the weight, communication time and computation time of each job are generated randomly, where weight ranges from 1 to 30, and communication time and computation time are set to the same order of magnitude (ranging from a few seconds to a dozen seconds) except the experiment to verify the impact of $\beta$ on WJCT. Besides, the iteration number of each job is also set randomly and usually varies from a dozen to a few hundred, but it will range from hundreds to hundreds of thousands in the experiment that tests the impact of iteration number.

**Impact of** $\beta$: The communication time and computation time are usually different for different models. To investigate the impact of the ratio of computation time to communication time $\beta$, we have carried out corresponding experiments for seven scenarios by adjusting the computation time of DML jobs. The results are shown in Fig. 9. On the whole, we can see that SWRTF outperforms the other schemes in all scenarios. This is because SWRTF takes full advantage of the interlacing nature of computation and communication stages of DML, and effectively uses idle circuits to transmit data of other jobs when current served job is in the computational phase, which greatly improve the circuit utilization. In particular, it is observed that the variation range of WJCT of SWRTF is much smaller than that of SJF and Baraat. For instance, when $\beta$ changes from 0.1 to 0.5, WJCTs of SJF and Baraat increase by 35.24% and 35.31%, respectively, while that of SWRTF only increases by 8.73%. Note that, the performance of WF is only 11.3% lower than that of SWRTF on average. This indicates that weight has a substantial impact on WJCT and considering weight is essential to reduce WJCT. Although the performance improvement of SWRTF relative to WF is not particularly remarkable in this experiment, in most cases, considering both RCT and weight can obtain much higher performance, as can be seen from the later experiments (see Figs. 10–16).

**Impact of iteration number**: Fig. 10 shows the curves of WJCT as the iteration numbers of DML jobs change. It should be noted that we zoom in or out the iteration numbers of all jobs simultaneously in this experiment. Theoretically, more iterations means more computation stages SWRTF can leverage to transfer data, which is confirmed by Fig. 10. SWRTF can reduce WJCT by 42.9%, 54.2%, 27.2% on average compared to SJF, Baraat and WF, respectively. As can be seen from the figure, when the iteration number increases from thousands to tens of thousands, the growth rate of SWRTF is significantly lower than Baraat, SJF and WF. In addition, when the iteration number increases from tens of thousands to hundreds of thousands, this phenomenon becomes more obvious.

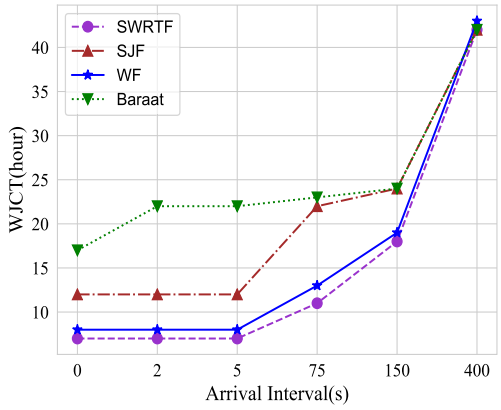


**Fig. 11.** WJCT Comparison with different job arrival intervals.

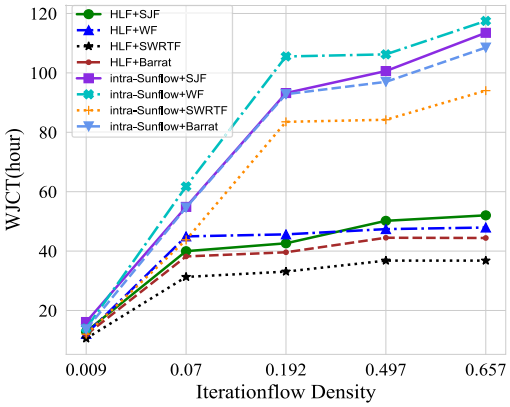


**Fig. 13.** WJCT Comparison with different *iterationflow densities*.

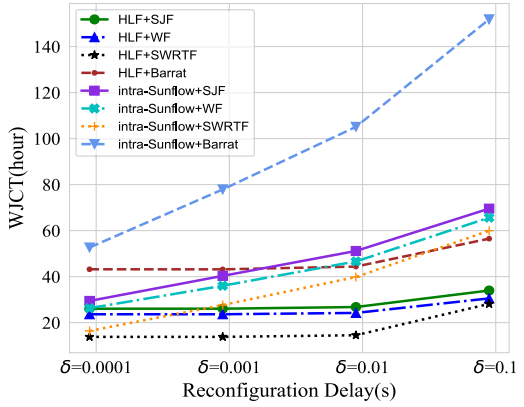


**Fig. 12.** WJCT Comparison with different reconfiguration delays.

**Impact of job arrival interval**: To test the online performance of SWRTF, we run a series of experiments with different job arrival intervals, and the simulation results are shown in Fig. 11. Notably, SWRTF always achieve the lowest WJCT. As the arrival interval increases, the gap between the four algorithms decrease, and this gap is even close to 0 when the arrival interval is 400 s. This is because when the interval is large enough, even longer than the completion time of a job, the jobs are executed sequentially. At this point, the results of any scheduling algorithms will be the same.

### 5.3. Comparison of intra-job+inter-job scheduling

In order to evaluate the performance of combined HLF and SWRTF, we compare the WJCT obtained by combining different intra-job and inter-job scheduling algorithms.

In this part, the experiment details are the similar to that of the previous two sections. We also use VGG-16 (528 MB) and unless otherwise specified, the circuit bandwidth is set to 100 Gbps and reconfiguration delay is 0.01 s. Besides, the iteration number of each job is set to tens to hundreds, and the computation time also ranges from a few seconds to a few tens of seconds.

**Impact of reconfiguration delay**: Fig. 12 shows the performance of combining different intra-job and inter-job algorithms. Similar to Fig. 7, as reconfiguration delay increases, the result also increases due to longer circuit switch time. Besides, our algorithms (HLF+SWRTF) always achieve the lowest WJCT. Compared with intra-Sunflow+SWRTF, our algorithms take about 15.8% less time when $\delta$ is 0.0001 s, but is 45.6% lower on average. This is reasonable because when $\delta$ is lower, the circuit switch time is a smaller part of communication time, and the gap between HLF and intra-Sunflow will be smaller. Besides, when $\delta$ is 0.1 s, our algorithms achieve only 7.8% better performance than HLF+WF, but they can get 32.76% less WJCT on average. Like Fig. 9, this also indicates that the weight is an essential factor to obtain a small WJCT. Similarly, in most cases of this experiment, a combination of RCT and weight can yield much greater performance gains.

**Impact of *iterationflow density*:** It can be seen from Fig. 13 that as *iterationflow density* $r$ grows, WJCT also grows due to longer communication time of each job (see Fig. 8), and our algorithms also have the lowest WJCT. Besides, the performance of algorithms using HLF is significantly better than that using intra-Sunflow, especially when $r$ is bigger, which indicates the effectiveness of HLF.

**Impact of circuit bandwidth**: The impact of bandwidth on WJCT is shown in Fig. 14. In general, WJCT decreases as bandwidth grows, due to less communication time. Besides, WJCT of our algorithms is always the lowest compared to other algorithms. For example, the WJCT of our algorithms is 45.48% and 36.56% lower than HLF+WF and intra-Sunflow+SWRTF on average, respectively.

**Impact of iteration number**: The influence of iteration number of combined algorithms on WJCT is shown in Fig. 15. Similar to Fig. 10, all WJCTs increase as iteration number increases because of a longer job completion time for each job. It also indicates that our algorithms are superior to the others thanks to the effectiveness of HLF and SWRTF. For example, the performance of our algorithms is 41.68%, 57.66%, 37.35% better than that of intra-Sunflow+SWRTF, intra-Sunflow+WF and HLF+WF on average.

**Impact of $\beta$**: After getting the communication time of each job through HLF or intra-Sunflow, we test the impact of $\beta$ on WJCT by adjusting the computation time, the results are shown in Fig. 16. With the increase of $\beta$, WJCT grows significantly because of longer job completion time, and our algorithms always achieve the lowest WJCT. Besides, the performance of HLF+SWRTF is 34.98% better than HLF+WF on average, which is much higher than that of Fig. 9. This also indicates that for the jobs in this experiment, joint consideration of RCT and weight can achieve better performance. Compared with intra-Sunflow+SWRTF, our algorithms can obtain about 44.79% less WJCT on average.

**Impact of job arrival interval**: Fig. 17 compares the impact of job arrival interval on WJCT. In general, the change trend of WJCT is the same as that of Fig. 11, and all results will be the same when arrival interval is large enough. Note that, when arrival interval is 50 s, WJCT of HLF+SJF is a little bigger than that of intra-Sunflow+SJF. This demonstrates that for SJF, if the communication time of each job becomes smaller, it may not be possible to obtain

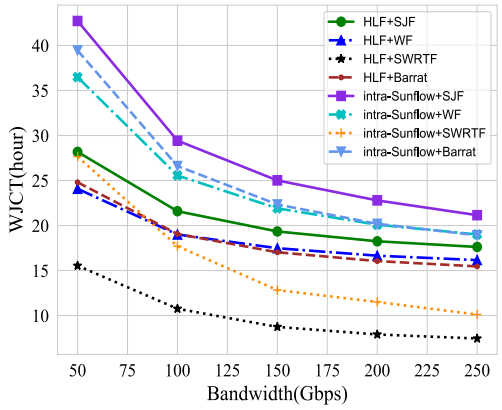


**Fig. 14.** WJCT Comparison with different bandwidths.

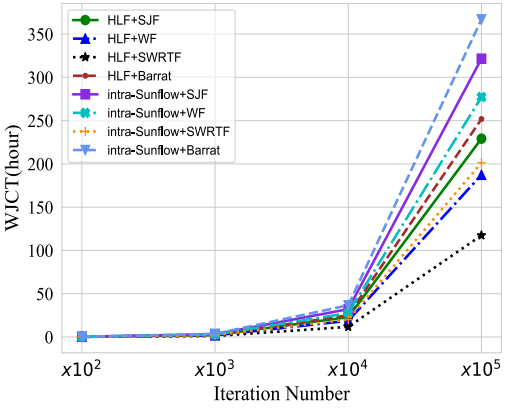


**Fig. 15.** WJCT Comparison with different iteration numbers.

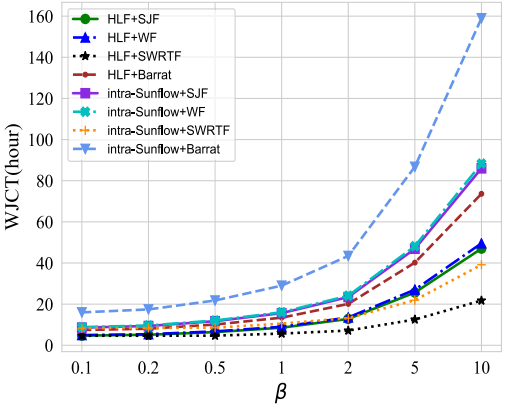


**Fig. 16.** WJCT Comparison with different ratios of the computation time to communication time $\beta$.

a smaller WJCT. At the same time, it also indicates that it is not enough to only consider *RCT* and ignore weight.

## 6. Related work

**DML scheduling in packet-switched networks**. Recently, many scheduling schemes for DML have emerged in packet-switched networks. FlexPS [43] is a new PS system which allows to change *parallelism degree* in multiple stages during runtime of a machine learning task. Cicada [44] exploits the Shortest-Job-First (SJF) to schedule coflows of DML jobs based on the

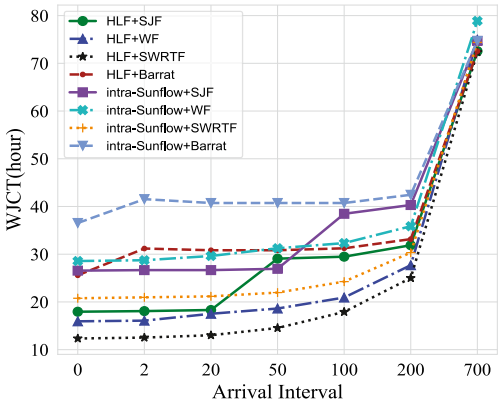


**Fig. 17.** WJCT Comparison with different job arrival intervals.

characteristic of coflow self-similarity of DML. In order to mitigate the straggler effect for DML, [45] presents TensorLights, a traffic scheduler at end-host NICs, whose key idea is to apply the same priority to the flows of the same job. TicTac [46] aims to schedule the *ops* order in DAG for the purpose of better overlap of communication and computation in TensorFlow, resulting in less iteration time. Jayarajan et al. [8] propose Priority-based Parameter Propagation (P3), a new parameter synchronization mechanism for PS architecture. P3 synchronizes parameters at a fine granularity, where layers are split into smaller slices and these slices are independently scheduled according to their priorities. Furthermore, Peng et al. [9] present ByteScheduler, a generic communication scheduler for distributed DNN training through combining priority-based communication with tensor partitioning based on Bayesian Optimization. Unlike these schemes, our algorithm does not care about the parameter priority at the end servers, but considers a novel OCS network and the specific flow scheduling within the network to reduce WJCT from a different perspective.

**Circuit scheduling**. The optical circuit scheduling algorithms can be divided into flow-based scheduling [47–49] and coflow-based scheduling [18–20]. In general, the former approaches are usually used for hybrid circuit/packet networks. Solstice [47] is a heuristic algorithm based on Birkoff–von Neumann (BvN), and the evaluation results show it can increase circuit utilization significantly. [49] presents two greedy algorithms, Eclipse for direct routing and Eclipse++ for indirect routing, and the simulation results demonstrate Eclipse achieves better performance than [47] and BvN. Meanwhile, Eclipse++ is always better than Eclipse in different scenarios. In BFF [48], maximum weighted matching (MWM) is first executed, and once an output/input port is idle, it immediately finds another input/output port with the longest processing time, while our algorithm (HLF) firstly serves the heaviest load port of all free ports. Sunflow [18] is the first coflow scheduling approach in circuit networks, where flows in a coflow are scheduled in arbitrary order and the coflows are scheduled in priority order. A joint shaping of circuit configuration and application's traffic demand is proposed in [20], and it can effectively reduce the CCT according to its evaluation results. Wang et al. [19] define the integrated coflow and circuit scheduling (GCCS) problem to minimize the CCT, where the routing through multiple OCSes and circuit configuration are considered. Besides, extensive simulations show the integrating algorithm and lower-complexity algorithm GCCS can improve the performance by 43%–70% and 37%–53%, respectively.

The above researches only consider how to minimize CCT or traffic completion time. However, DML job contains multi-stage communication stages and computation stages, which is

completely different from the application demands of the above works. Therefore, the circuit scheduling algorithm for DML jobs deserves to be studied.

## 7. Conclusion

In this paper, we firstly introduce the problem of DML scheduling in circuit-switched networks, and then propose intra-job scheduling algorithm HLF and inter-job scheduling algorithm SWRTF based on the characteristics of DML. To validate our design, we conduct large-scale simulations in different cases, and the results have shown that our algorithms can efficiently improve the DML job training performance. Specifically, experiments also demonstrate that HLF has more stable performance than Sunflow in terms of different *iterationflow* density and reconfiguration delay. However, one drawback of HLF is a higher time complexity. Therefore, an approach with lower time complexity is the future work we are interested. Besides, we also assume each worker has the same computation capability, without consideration of the straggler problem. Thus, heterogeneous workers will be considered in our future research.

## CRediT authorship contribution statement

**Ling Liu:** Conceptualization, Methodology, Software, Writing - original draft, Validation. **Hongfang Yu:** Supervision, Funding acquisition, Project administration. **Gang Sun:** Supervision, Validation, Formal analysis, Writing - review & editing. **Huaman Zhou:** Investigation, Writing - review & editing. **Zonghang Li:** Writing - review & editing. **Shouxi Luo:** Writing - review & editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This research was partially supported by the National Key Research and Development Program of China (2019YFB1802800), PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications (PCL2018KP001).

## References

[1] E.P. Xing, Q. Ho, P. Xie, D. Wei, Strategies and principles of distributed machine learning on big data, Engineering. 2 (2) (2016) 179–195.
[2] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, et al., Gandiva: Introspective cluster scheduling for deep learning, in: 13th {USENIX} Symposium on Operating Systems Design and Implementation, OSDI 18, CARLSBAD, USA, 2018, pp. 595–610.
[3] M. Alan, A. Panda, D. Bottini, L. Jian, P. Kumar, S. Shenker, Network evolution for dnns, SysML 1 (2018) 1–3, doc/182.
[4] L. Luo, J. Nelson, L. Ceze, A. Phanishayee, A. Krishnamurthy, Parameter hub: a rack-scale parameter server for distributed deep neural network training, in: The ACM Symposium on Cloud Computing, Carlsbad, California, 2018, pp. 41–54.
[5] J.H. Park, S. Kim, J. Lee, M. Jeon, S.H. Noh, Accelerated training for CNN distributed deep learning through automatic resource-aware layer placement, 2019, arXiv preprint arXiv:1901.05803.
[6] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, E.P. Xing, Poseidon: An efficient communication architecture for distributed deep learning on {GPU} clusters, in: 2017 {USENIX} Annual Technical Conference, USENIX 17, Santa Clara, CA, 2017.
[7] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Guo, Optimus: an efficient dynamic resource scheduler for deep learning clusters, in: Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 2018.
[8] A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, G. Pekhimenko, Priority-based parameter propagation for distributed DNN training, 2019, arXiv preprint arXiv:1905.03960.
[9] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, C. Guo, A generic communication scheduler for distributed DNN training acceleration, in: Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019, pp. 16–29.
[10] S. Wang, D. Li, J. Geng, Y. Gu, Y. Cheng, Impact of network topology on the performance of DML: Theoretical analysis and practical factors, in: IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, 2019, pp. 1729–1737.
[11] L. Liu, Q. Jin, D. Wang, H. Yu, G. Sun, S. Luo, Psnet: Reconfigurable network topology design for accelerating parameter server architecture based distributed machine learning, Future Gener. Comput. Syst. 106 (2020) 320–332.
[12] W.M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A.C. Snoeren, G. Porter, Rotornet: A scalable, low-complexity, optical datacenter network, in: The Conference of the ACM Special Interest Group on Data Communication, Los Angeles, CA, USA, 2017, pp. 267–280.
[13] Q. Cheng, M. Bahadori, M. Glick, S. Rumley, K. Bergman, Recent advances in optical technologies for data centers: a review, Optica 5 (11) (2018) 1354–1370.
[14] L. Luo, K.-T. Foerster, S. Schmid, H. Yu, Deadline-aware multicast transfers in software-defined optical wide-area networks, IEEE J. Sel. Areas Commun. (2020) 1–16.
[15] N. Farrington, G. Porter, S. Radhakrishnan, H.H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, Helios: a hybrid electrical/optical switch architecture for modular data centers, ACM SIGCOMM Comput. Commun. Rev. 41 (4) (2011) 339–350.
[16] G. Wang, D.G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, M. Ryan, C-through: Part-time optics in data centers, ACM SIGCOMM Comput. Commun. Rev. 41 (4) (2011) 327–338.
[17] Y. Xiaoshan, X. Hong, G. Huaxi, L. Hao, THOR: A scalable hybrid switching architecture for data centers, IEEE Trans. Commun. 66 (10) (2018) 4653–4665.
[18] X.S. Huang, X.S. Sun, T. Ng, Sunflow: Efficient optical circuit scheduling for coflows, in: The 12th International on Conference on Emerging Networking EXperiments and Technologies, Irvine, CA, USA, 2016, pp. 297–311.
[19] H. Wang, X. Yu, H. Xu, J. Fan, C. Qiao, L. Huang, Integrating coflow and circuit scheduling for optical networks, IEEE Trans. Parallel Distrib. Syst. 30 (6) (2018) 1346–1358.
[20] H. Zhang, K. Chen, M. Chowdhury, Pas de deux: Shape the circuits, and shape the apps too!, in: The 2nd Asia-Pacific Workshop on Networking, Beijing, China, 2018, pp. 29–35.
[21] B. Tian, C. Tian, H. Dai, B. Wang, Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time, in: IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, 2018, pp. 864–872.
[22] P. Zhou, H. Yu, G. Sun, Grouper: Accelerating hyperparameter searching in deep learning clusters with network scheduling, IEEE Trans. Netw. Serv. Manag. (2020) 1–17.
[23] F.R. Dogar, T. Karagiannis, H. Ballani, A. Rowstron, Decentralized task-aware scheduling for data center networks, ACM SIGCOMM Comput. Commun. Rev. 44 (4) (2014) 431–442.
[24] E. Honda, Y. Mori, H. Hasegawa, K.-I. Sato, Feasibility test of large-scale (1,424× 1,424) optical circuit switches utilizing commercially available tunable lasers, in: 2019 24th OptoElectronics and Communications Conference (OECC) and 2019 International Conference on Photonics in Switching and Computing (PSC), IEEE, 2019, pp. 1–3.
[25] T.J. Seok, K. Kwon, J. Henriksson, J. Luo, M.C. Wu, Wafer-scale silicon photonic switches beyond die size limit, Optica 6 (4) (2019) 490–494.
[26] N. Dupuis, J.E. Proesel, N. Boyer, H. Ainspan, C.W. Baks, F. Doany, E. Cyr, B.G. Lee, An 8× 8 silicon photonic switch module with nanosecond-scale reconfigurability, in: Optical Fiber Communication Conference, Optical Society of America, 2020, pp. Th4A–6.
[27] K. Ikeda, K. Suzuki, R. Konoike, S. Namiki, H. Kawashima, Large-scale silicon photonics switch based on 45-nm CMOS technology, Opt. Commun. 466 (2020) 1–7.
[28] Y. Xia, X.S. Sun, S. Dzinamarira, D. Wu, X.S. Huang, T. Ng, A tale of two topologies: Exploring convertible data center network architectures with flat-tree, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, Angeles, CA, USA, 2017.
[29] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, ACM SIGCOMM Comput. Commun. Rev. 38 (4) (2008) 63–74.
[30] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, et al., Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network, ACM SIGCOMM Comput. Commun. Rev. 45 (4) (2015) 183–197.

[31] F. Sattler, S. Wiedemann, K.-R. Müller, W. Samek, Sparse binary compression: Towards distributed deep learning with minimal communication, in: IEEE International Joint Conference on Neural Networks, IJCNN, 2019, Budapest, Hungary, 2019, pp. 1–12.
[32] J. Liu, B. Zhuang, Z. Zhuang, Y. Guo, J. Huang, J. Zhu, M. Tan, Discrimination-aware network pruning for deep model compression, 2020, arXiv preprint arXiv:2001.01050.
[33] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, in: LearningSys NIPS, 2015, pp. 1–6.
[34] T. Ben-Nun, T. Hoefler, Demystifying parallel and distributed deep learning: An in-depth concurrency analysis, ACM Comput. Surv. 52 (4) (2019) 1–43.
[35] O. Hartmann, M. Kühnemann, T. Rauber, G. Rünger, Adaptive selection of communication methods to optimize collective MPI operations, in: Workshop on Compilers for Parallel Computers, CPC, A Coruna, Spain, 2006.
[36] P. Patarasuk, X. Yuan, Bandwidth optimal all-reduce algorithms for clusters of workstations, J. Parallel Distrib. Comput. 69 (2) (2009) 117–124.
[37] R. Thakur, R. Rabenseifner, W. Gropp, Optimization of collective communication operations in MPICH, Int. J. High Perform. Comput. Appl. 19 (1) (2005) 49–66.
[38] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D.R. Ports, P. Richtárik, Scaling distributed machine learning with in-network aggregation, vol. 1, 2019, pp. 1–16, arXiv preprint arXiv:1903.06701.
[39] S. Zhang, S. Zhang, X. Zhang, Z. Qian, M. Xiao, J. Wu, J. Ge, X. Wang, Far-sighted multi-stage awasre coflow scheduling, in: 2018 IEEE Global Communications Conference, GLOBECOM, 2018, pp. 1–7.
[40] L. Luo, K.-T. Foerster, S. Schmid, H. Yu, Splitcast: Optimizing multicast flows in reconfigurable datacenter networks, in: 40th IEEE International Conference on Computer Communications, INFOCOM, Toronto, Canada, 2020.
[41] E.J. Anderson, C.N. Potts, On-line scheduling of a single machine to minimize total weighted completion time, in: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2002, pp. 548–557.
[42] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, K. Ramchandran, Speeding up distributed machine learning using codes, IEEE Trans. Inform. Theory 64 (3) (2017) 1514–1529.
[43] Y. Huang, T. Jin, Y. Wu, Z. Cai, X. Yan, F. Yang, J. Li, Y. Guo, J. Cheng, Flexps: Flexible parallelism control in parameter server architecture, Proc. VLDB Endow. 11 (5) (2018) 566–579.
[44] G. Yang, Y. Jiang, Q. Li, X. Jia, M. Xu, Cross-layer self-similar coflow scheduling for machine learning clusters, in: 27th International Conference on Computer Communication and Networks, ICCCN 2018, Hangzhou, China, 2018, pp. 1–9.
[45] X.S. Huang, A. Chen, T. Ng, Green, yellow, yield: End-host traffic scheduling for distributed deep learning with tensorlights, Update 3 (2019) 1–8.
[46] S.H. Hashemi, S.A. Jyothi, R.H. Campbell, TicTac: Accelerating distributed deep learning with communication scheduling, 2018, arXiv preprint arXiv:1803.03288.
[47] H. Liu, M.K. Mukerjee, C. Li, N. Feltman, G. Papen, S. Savage, S. Seshan, G.M. Voelker, D.G. Andersen, M. Kaminsky, et al., Scheduling techniques for hybrid circuit/packet networks, in: Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies, Heidelberg, Germany, 2015, pp. 1–13.
[48] L. Liu, L. Gong, S. Yang, J. Xu, L. Fortnow, Best first fit (BFF): An approach to partially reconfigurable hybrid circuit and packet switching, in: 2018 IEEE 11th International Conference on Cloud Computing, CLOUD, Seattle, WA, USA, 2018, pp. 426–433.
[49] S. Bojja Venkatakrishnan, M. Alizadeh, P. Viswanath, Costly circuits, submodular schedules and approximate carathéodory theorems, in: ACM SIGMETRICS Performance Evaluation Review, vol. 44, no. 1, 2016, pp. 75–88.