P4INC-AOI: All-Optical Interconnect Empowered by In-Network Computing for DML Workloads

Xuexia Xie, Binjun Tang, Xiaoliang Chen, and Zuqing Zhu, *Fellow, IEEE*

***Abstract*—Increasing demands for distributed machine learning (DML) have posed significant pressure on data-center networks (DCNs). This promotes the adoption of reconfigurable all-optical interconnects (AOI) in DCNs leveraging optical circuit switching (OCS) for better performance on throughput, energy efficiency, and data transfer latency. Despite their benefits, these OCS-based DCNs (ODCNs) still bear limited flexibility due to the larger switching granularity and longer reconfiguration latency of OCS. To address this issue, this work introduces in-network computing (INC) in an ODCN to realize P4INC-AOI, which can orchestrate INC and AOI to explore their mutual benefits for accelerating the training of DML jobs with less AOI reconfigurations. In the control plane of P4INC-AOI, we address the scheduling of concurrent DML jobs by formulating a mixed integer linear programming (MILP) model and proposing a time-efficient heuristic, to allocate multi-dimensional resources and configure AOI for minimizing the longest job completion time (JCT) across workloads. For the data plane, we extend existing in-network gradient aggregation schemes to accelerate DML jobs more efficiently. We first implement P4INC-AOI and verify its performance in a small-scale ODCN testbed, and further justify its effectiveness with large-scale simulations. Our experimental results demonstrate that compared with an ODCN without INC, P4INC-AOI not only cuts down AOI reconfigurations effectively but also reduces the average JCT of DML jobs in ResNet50 and VGG16 by 46.66% and 56.34%, respectively.**

***Index Terms*—All-optical interconnect, Data-center networks, P4, In-network computing, Distributed machine learning.**

I. Introduction

OVER the past decade, the fast development of artificial intelligence (AI) has fueled an explosive increase in the sizes of machine learning (ML) models and their training data sets [1, 2]. For instance, as one of the most famous large language models (LLMs), GPT-3 features approximately 175 billion trainable parameters, while the size of its training data set is $\sim$45 TB [3]. Consequently, distributed machine learning (DML), which fuses the computing and storage power of a number of servers outfitted with hardware accelerators (*e.g.*, graphics processing units (GPUs)) to make the training of large ML models tractable, becomes inevitable [4–6]. Specifically, people can form DML clusters in a data-center network (DCN) and leverage parallelization techniques such as data/model parallelism to accelerate ML training [7]. However, the synchronization and updating of parameters in each DML cluster can cause a huge volume of data transfer, posing great pressure on the architecture of DCNs. Moreover, recent studies have already shown that due to the insufficient inter-pod throughput in today's DCNs, the communication phase can more likely be the bottleneck of DML training [8–10].

Most of today's DCNs are still purely based on electrical packet switching (EPS), where inter-rack/pod communication goes through a hierarchical EPS network planned based on the worst-case scenario [11]. This, however, restricts the average resource usage and adaptivity of these DCNs [12–14]. Hence, although DCN operators have upgraded their EPS switches frequently, it is still challenging for EPS-based DCNs to cope with the ever-increasing traffic demands from bandwidth-intensive applications such as DML training [15]. Furthermore, EPS also possesses other intrinsic drawbacks, *e.g.*, high power consumption and long processing delay [16–18].

Therefore, researchers have tried to introduce optical circuit switching (OCS) into DCNs for better performance on throughput, data transfer latency, and energy efficiency [19–25]. Specifically, an OCS-based DCN (ODCN) replaces the spine layer in its EPS-based counterpart with an all-optical interconnect (AOI) built with OCS switches [26]. Then, the ODCN's topology can be reconfigured with topology engineering (TPE) to better adapt to dynamic and skewed traffic demands [15]. Although TPE effectively enhances the adaptivity of ODCNs, it can hardly be done without causing service interruptions, especially with the currently-deployed commercial OCS switches that bear fiber-level switching granularity and millisecond-scale reconfiguration latency [18, 26]. This issue offsets the benefits of ODCNs and makes it challenging for them to keep up with the fast development of large-scale DML training, especially when the DML workloads are highly dynamic [27]. While advanced photonic technologies can be utilized to achieve low-latency and fine-grained optical switching [28–30], the solutions either still stay in the laboratorial validation phase [30] or have difficulty to be commercialized due to scalability limitations [28] or high capital costs [29]. Consequently, ODCNs built on commercial spatial OCS switches present the mainstream choices of current DCs [31].

On the other hand, the impact of long reconfiguration latency can be mitigated if we can reduce the frequency of TPE but still keep the match degree between an ODCN's topology and traffic matrix at a relatively high level. However, this can hardly be done if we cannot make the traffic matrix malleable [32]. The advances on programmable data plane (PDP) have revealed that a PDP-based EPS switch can process packets with arithmetic operations according to customized protocols and network functions [33]. Hence, PDP empowers networks with in-network computing (INC) [34], which reshapes or even terminates traffic flows as they traverse a network. This enables us to constantly fit the topology of an ODCN to its current

X. Xie, B. Tang, X. Chen and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: xlichen@ieee.org).

traffic matrix, and thereby, to implicitly reduce the frequency of AOI reconfigurations [35]. Specifically, we can deploy PDP switches at top-of-rack (ToR) or egress points of pods in an ODCN to largely confine data transfers within racks/pods with distributed INC, which otherwise would consume AOI connections. Consequently, this allows for orchestrating clusters of workloads in the ODCN to minimize their job completion time (JCT) while reducing unnecessary AOI reconfigurations.

Previously, in [35], we proposed P4INC-AOI to implement the aforementioned idea of orchestrating INC and AOI to accelerate distributed computing workloads in an ODCN, and used Hadoop MapReduce jobs as an example to demonstrate the effectiveness of P4INC-AOI. To the best of our knowledge, it was the first attempt to explore the mutual benefits of INC and AOI for distributed computing workloads in ODCNs.

However, the study in [35] was still preliminary for the following reasons. First, it did not design an algorithm to schedule dynamic computing clusters in P4INC-AOI. Note that, the algorithm design is not trivial as it needs to jointly consider AOI reconfigurations and the unique features of INC. Specifically, INC can reshape the traffic generated by computing clusters, which results in a malleable traffic matrix and makes existing job scheduling schemes (*e.g.*, those in [36, 37]) inapplicable. Moreover, INC introduces a new type of resources (*i.e.*, memory in PDP switches) to the job scheduling problem, whose allocation is critical for reducing JCT. Second, the data plane implementation in [35] was very preliminary, since it did not enable INC for DML training or address the practical issues when INC needs to handle multiple dynamic computing clusters (*e.g.*, cluster indexing, traffic congestion, and competition on PDP switch resources).

In this work, we extensively expand P4INC-AOI to make it suitable for orchestrating INC and AOI to accelerate dynamic DML workloads with less AOI reconfigurations. We first model an ODCN as a discrete-time system and formulate a mixed integer linear programming (MILP) model to jointly tackle the allocation of multi-dimensional resources (*i.e.*, IT resources in server pools, memory in PDP switches, and bandwidth on inter-rack/pod links) and configuration of AOI to minimize the longest JCT across DML workloads. Then, we prove the problem of scheduling dynamic DML workloads in P4INC-AOI is $\mathcal{NP}$-hard, and propose a time-efficient heuristic that can determine whether each DML cluster should be served with INC and how to schedule them based on the remaining multi-dimensional resources in the ODCN and the impact on in-service DML workloads. Next, we extend the existing data plane implementations [38, 39] to enable INC to handle dynamic DML workloads in P4INC-AOI. Finally, we conduct extensive numerical simulations to evaluate our proposal and also test it experimentally in a small-scale ODCN testbed that is built with off-the-shelf components. Both simulation and experimental results verify the effectiveness of our proposal.

Our major contributions can be summarized as follows:

- To the best of our knowledge, this is the first study on how to orchestrate INC and AOI to schedule dynamic DML workloads in an ODCN.
- We formulate an MILP model to describe the optimization for minimizing the longest JCT across DML workloads in P4INC-AOI, and design a time-efficient heuristic that can be practically implemented in the control plane.
- We design and implement the data plane of P4INC-AOI to enable INC for dynamic DML workloads with PDP switches based on Tofino ASICs, and make sure that they work seamlessly with AOI reconfigurations.
- Both simulation and experimental results confirm that our proposed P4INC-AOI effectively accelerates DML training, and outperforms the ODCN without INC in training throughput, JCT, and AOI reconfigurations.

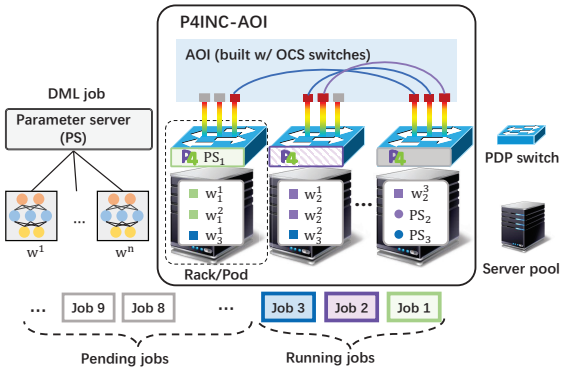


Fig. 1. Operation principle of P4AOI-INC, where the cluster handles multiple jobs, each with several workers. Workers from the same jobs are marked in the same colors, and w_i^j denotes the j-th worker of job i.

The rest of the paper is organized as follows. Section II discusses the related work. We present the operation principle of P4INC-AOI in Section III. The MILP model and heuristic for scheduling DML workloads in P4INC-AOI are described in Sections IV and V, respectively. In Section VI, we explain the system design and implementation of the data plane of P4INC-AOI. The performance evaluations are discussed in Section VII. Finally, Section VIII summarizes this paper.

II. Related Work

In this section, we review the basics about DML, explain the opportunities and challenges of training DML in ODCNs, and comment on INC and its applications for DML acceleration.

A. DML Training

DML realizes parallelization by dividing its training workload among multiple computing nodes (namely, workers) to relieve the pressure on computing and storage power of each worker [4]. One common approach is data parallelism, which splits a large training data set across workers and lets each worker train an ML model over its portion to obtain a *local gradient* [7]. The local gradients are then combined to update the *global gradient* using an algorithm like the stochastic gradient descent (SGD) [40]. This procedure is repeated in iterations for model convergence and parameter optimization [41], and a number of communication frameworks have been developed for the synchronization and updating of parameters among workers, such as the parameter server, ring-AllReduce, distributed data parallel, and peer-to-peer [42].

Among the communication frameworks, the parameter server is a high-profile setup that organizes one or more parameter

servers (PS') and several workers in a tree-like cluster [4, 10]. Each worker sends its local gradients to a PS (*i.e.*, incast). Then, the PS obtains global gradients by averaging the local gradients and updates the parameters of the global model to the workers for further computation (*i.e.*, broadcast). While PS facilitates fast gradient aggregation with fewer steps, each training iteration in PS involves collective communications between the PS and workers with incast and broadcast, which can consume huge bandwidth [43]. Consequently, today's DML training widely adopts ring-based [44] or hybrid parallelism schemes [45], which are more scalable and less vulnerable to bandwidth bottlenecks. Nevertheless, more recent studies have revealed that INC-based traffic aggregation and reshaping can effectively mitigate the bottlenecks caused by PS and promote its state-of-the-art performance in DML training [10, 38]. The benefit of INC when combined with hybrid parallelism schemes remains a relevant yet under-explored area.

B. DML Training in ODCNs

As we have explained above, the collective communications caused by DML training can generate bottlenecks in a traditional DCN that uses a hierarchical EPS-based inter-rack/pod network. This issue can potentially be resolved by considering ODCNs, whose advantages span throughput, data transfer latency, energy efficiency, and topological flexibility [26]. Since the aggregated traffic of DML training at the pod level is highly predictable and exhibits recurring spatial patterns, most of today's ODCN approaches consider to replace the spine layer of EPS-based DCNs with an inter-pod AOI built with OCS switches [15, 31, 46, 47], to adapt to slow inter-pod traffic changes with low-frequency AOI reconfigurations. However, the millisecond-level reconfiguration latency of the commercial OCS switches used in these ODCNs can still induce troublesome service interruptions [27] considering the millisecond-scale dynamics of collective communications [48] and the coexistence of heterogeneous DML jobs. On the other hand, by incorporating the OCS technologies that can deliver a reconfiguration latency at microsecond or even nanosecond level, researchers also tried to directly connect ToR switches to AOI [28, 29], hoping that high-frequency AOI reconfigurations can be invoked in respond to fast rack-level traffic changes. Nevertheless, these OCS technologies either can not support high port-count or are not ready for commercialization.

In all, the granularity and reconfiguration latency of OCS are still the major challenges for using ODCNs to support highly-dynamic DML workloads efficiently. Although continuing to reduce OCS reconfiguration latency can eventually resolve the challenges, how to realize it in a practical and commercially-available way is still an open research problem.

C. INC and its Applications for DML Training

INC aims to intercept and process traffic along its forwarding path with PDP hardware, including FPGA [49], SmartNIC [50], and programmable ASIC [51], and it offers significant advantages in reducing traffic volume in networks and shortening processing time at end-hosts [34]. Specifically, the study in [52] suggested that having a small portion of network devices support INC can lead to a significant reduction in network utilization. As for accelerating DML, various INC schemes have been proposed in the literature [38, 39, 53, 54], which all considered the parameter server setup. SwitchML [38] considered the case where all the workers of a DML job are located in the same rack, and proposed to leverage INC to offload the gradient aggregation of workers to the ToR switch. ATP [39] tried to use INC to accelerate multiple DML jobs with in-network aggregation, and designed a decentralized aggregator allocation mechanism. Panama [53] developed an FPGA-based in-network aggregation scheme and proposed a load-balancing protocol for it. GRID [54] studied gradient routing for DML training with in-network aggregation.

However, all the existing studies on INC for DML training assumed that EPS-based DCNs were used. Therefore, the mutual benefits of INC and AOI still have not been explored for accelerating DML training. Therefore, in this work, we design P4INC-AOI to study how to empower ODCN with INC for serving highly-dynamic DML workloads efficiently.

III. Operation Principle of P4AOI-INC

Fig. 1 explains the operation principle of our P4AOI-INC, where an AOI interconnects a set of network units, each of which consists of a PDP switch and a server pool. Here, we hope to point out that P4AOI-INC is generic enough such that each network unit can refer to either a rack or a pod. Specifically, if the network unit refers to a rack, the PDP switch is just the ToR switch, and if it represents a pod, the PDP switch is a virtual one, which is abstracted by considering all the EPS switches in the pod. The PDP switch enables INC for accelerating DML jobs, *i.e.*, we can aggregate gradients of a DML job in it. For jobs whose workers are within the same unit, INC can completely replace the PS for global gradient aggregation (*e.g.*, *Job* 1 in Fig. 1). For the case when the workers of a job are in different units, INC only aggregates the gradients from the workers in its unit and sends the partial results to the PS, while the rest of the aggregation operations follow a traditional scheme (*e.g.*, *Job* 2 in Fig. 1).

In addition to the PDP switches, we can also deploy the PS in a server pool, which means that INC is not turned on for the DML job (*e.g.*, the PS of *Job* 3 in Fig. 1). The workers of all the DML jobs can only be deployed in server pools, and necessary connections need to be set up to support the collective communications between each PS and its workers. For a DML job, if its PS and workers are in the same network unit, its communications only consume the uplink and downlink bandwidth between the corresponding server pool and PDP switch, and otherwise, we need to configure the AOI to bridge the induced cross-network-unit communications.

Note that, according to the analysis done by Intel [55], PDP switches are not more expensive or more power-consuming than legacy switches when running at the same data-rate. Hence, as offloading PS' to them can effectively reshape incast traffic and mitigate bandwidth bottlenecks, which otherwise would require installation of more network interface cards and transceiver modules to sustain the same performance level, PDP switches actually makes ODCNs more cost-effective [56].

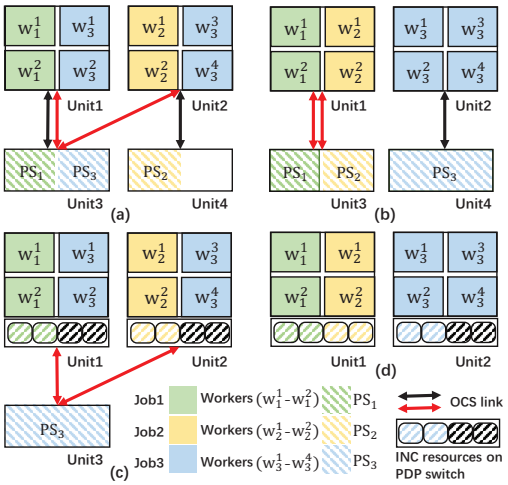


Fig. 2. Examples to show mutual benefits of INC and AOI for DML training, (a)-(b) schemes without INC, and (c)-(d) schemes with INC. Here, w_i^j still denotes the j-th worker of job i. Resources occupied by the same jobs are marked in the same colors. For instance, green is for *Job* 1: in (a), *Job* 1 uses the CPU resources in *Unit* 3 (PS_1), and in (c), *Job* 1 consumes the INC resources in its local unit. The units are interconnected via OCS links labeled by solid lines, with the red ones representing heavily-loaded links.

To understand the mutual benefits of INC and AOI for DML training, we plot four examples in Fig. 2. Here, we consider an ODCN that consists of four network units, where *Units* 1-2 are dedicated to GPU-based computation and can carry workers of DML jobs, while *Units* 3-4 are for CPU-based computation and can be used to instantiate PS'. There are three DML jobs to schedule, where *Job* 1 (green) and *Job* 2 (yellow) both use two workers, and *Job* 3 (blue) includes four workers. Figs. 2(a) and 2(b) show the schemes without INC.

In Fig. 2(a), four connections need to be set up through the AOI to support the cross-unit communications. As each DML job requires at least one new connection to bridge their workers and PS, three AOI reconfigurations are needed in the worst case, which can cause increased port utilization and traffic interruption. Additionally, units with multiple PS' (*e.g.*, *Unit* 3) may experience significant incast pressure (indicated by red lines). The scheme in Fig. 2(b) lets the workers run in a less fragmented way, *i.e.*, the workers of *Jobs* 1 and 2 all run in *Unit* 1 with their PS' placed in *Unit* 3, and the workers and PS of *Job* 3 are in *Units* 2 and 4, respectively. Then, we only need to set up three connections through the AOI, causing two AOI reconfigurations at most.

Note that, due to the dynamic arrival of workloads and fragmented server usage, a DCN operator cannot always place jobs optimally. Hence, the worker distributions shown in Figs. 2(a) and 2(b) can both happen in an ODCN. Figs. 2(c) and 2(d) illustrate the schemes with INC. In Fig. 2(c), the workers are distributed the same as that in Fig. 2(a), but with INC, we can offload the PS' of *Jobs* 1 and 2 to PDP switches, and only one PS need to be placed in *Unit* 3 for *Job* 3 to aggregate the gradients from *Units* 1 and 2. Then, INC reduces the number of connections through the AOI to two, and only one AOI reconfiguration is required as the connections are both for *Job* 3. Finally, Fig. 2(d) shows how to improve the scheduling scheme in Fig. 2(b) with INC. This time, the PS' of all the jobs are offloaded to PDP switches, and thus no connection needs to be set up through the AOI, *i.e.*, the three jobs can be served without requiring any AOI reconfiguration.

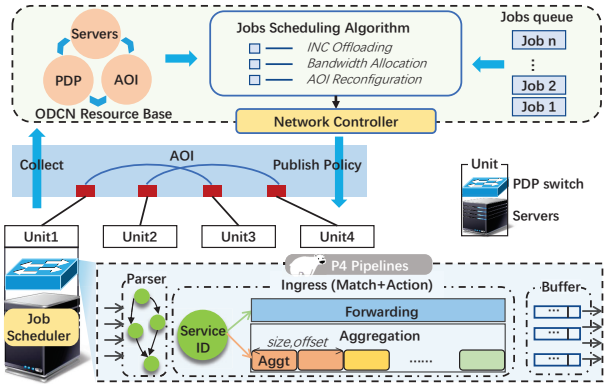


Fig. 3. System design.

Although the examples in Fig. 2 demonstrate the mutual benefits of INC and AOI for DML training, these benefits cannot be fully exploited without a carefully designed mechanism that can jointly optimize the allocation of multi-dimensional resources in the ODCN and the configuration of its AOI. Therefore, we design the system architecture in Fig. 3 for P4INC-AOI. The data plane follows the generic architecture of an ODCN, where each unit contains a PDP switch and multiple servers. The servers provide GPU resources for workers or CPU resources for PS'. The PDP switches enable INC for the synchronous gradient aggregation of DML jobs while retaining data forwarding functionalities. The control plane manages the job queue and monitors the ODCN's state. It orchestrates the multi-dimensional data plane resources by invoking the optimization to compute AOI reconfiguration, job scheduling and bandwidth allocation schemes. The operation principles of the key components in P4INC-AOI are elaborated as follows.

AOI: after job scheduling and traffic routing have been determined, the network controller configures the connections in AOI to provision inter-unit traffic.

Job schedulers: each server pool has a job scheduler that receives job scheduling schemes from the network controller, assigns a service ID to each job, and allocates to each INC job an aggregator segment in the PDP switch. When a job is scheduled to start, the job scheduler instructs its PS and workers to start the current training iteration.

PDP switches: PDP switches primarily undertake two tasks: aggregating gradients for INC jobs and data forwarding for no-INC jobs. Each switch identifies the type of a job using the service ID encoded in packet headers. To facilitate gradient aggregation, we abstract switch memory as a unified array of aggregators, label each aggregator segment by its size and offset, and assign each INC job to a segment in runtime. For non-INC jobs, the network controller installs table entries in each switch to steer gradient flows between PS' and workers.

IV. DML Workload Scheduling in P4INC-AOI

In this section, we first explain the network model of P4INC-AOI and formulate an MILP to describe the optimization for scheduling DML jobs with INC in it.

A. Network Model

For P4INC-AOI, we consider an ODCN that interconnects N network units, each of which consists of a PDP switch and a server pool. Hence, we can abstract each network unit as a virtual node $v \in V$, where V is the set of network units and we have $|V| = N$. Each node $v \in V$ connects to the AOI (*i.e.*, one or a set of commercial optical cross-connect (OXC) [15]) in the ODCN through $P_{\text{oxc},v}$ optical ports, each of which has a capacity of B_{port}. Each port can only connect to one other port through the AOI, and for simplicity, we assume that duplex communication can be realized through each pair of ports[1]. In other words, the wavelength(s) emitted from an optical port is/are switched from an input to an output port of an OXC as an ensemble. In each node v, we refer to the direction from the PDP switch to the server pool as *downlink*, while the reverse one is *uplink*. The uplink/downlink capacities of each PDP switch are equal, and their sum defines the bandwidth capacity of the switch, denoted as $B_{\text{PDP},v}$.

P4INC-AOI can enable INC in a PDP switch to facilitate the gradient aggregation for DML workers running in the server pool under the PDP switch. Specifically, for the workers of a same DML job, INC replaces their PS to calculate and return aggregation results to them, and in the process, INC consumes memory resources in PDP switches [38, 39]. Hence, we need to map the PS' of DML jobs to PDP switches if the jobs will be handled with INC, which is part of the scheduling problem considered in this work. We denote the INC capacity of a PDP switch $v \in V$ as $N_{\text{INC},v}$, in terms of the number of DML jobs.

For each DML job in P4INC-AOI, we assume that its PS and workers are placed by its service provider (*i.e.*, a different entity from the DCN operator). Therefore, the placement of the PS and workers of each DML job is the input to our DML job scheduling problem, and without loss of generality, we assume that the PS and workers of a DML job can be placed in either the same unit or different ones. As explained in Section II-A, there are four key steps in each training iteration of a DML job, *i.e.*, local calculation, pull gradients, parameter update, and push gradients. The local calculation is performed by the workers and its duration is a constant for each DML job, and thus the optimization of DML job scheduling does not need to consider it. As for the parameter update, its duration depends on where it is performed (*i.e.*, in the PS placed in a server pool or a PDP switch with INC) and the number of DML model's parameters, and thus the duration can be treated as a constant. Note that, for the case with INC, the duration of parameter update can be assumed to be zero because PDP switches perform gradient aggregation in a pipelined manner [38, 39] (*i.e.*, the parameter update is made concurrent with the steps of pull/push gradients). Finally, the durations of pull/push gradients rely on the bandwidth allocated to the communications between the workers and PS, which will be determined by our DML job scheduling.

Based on the considerations above, the DML job scheduling in P4INC-AOI becomes: given a batch of concurrent DML jobs and the locations of their PS' and workers, we optimize which jobs will be processed with INC, when the communications of the pull/push gradients of each job should start, and how to allocate bandwidth to the communications, such that the completion time of a batch training iteration (CT-BTI) of the DML jobs can be minimized.

Definition 1. *We define the* CT-BTI *of a batch of concurrent DML jobs as the duration from when the first job starts pulling gradients to when the last job finishes its parameter update, for one training iteration of all the jobs.*

Note that, the ultimate goal of the DML job scheduling should be to minimize the longest JCT of DML jobs, for effectively accelerating their training. As we schedule a batch of concurrent jobs together, this ultimate goal can be approximated as to minimize the CT-BTI of the jobs if we consider a reasonable assumption that each job will be trained in a relatively large number of iterations and the total training iterations of all the jobs are very similar. Moreover, we only need to consider the durations of pull gradients and parameter update for each job in the CT-BTI for the following reason. As the steps of pull/push gradients are symmetric in terms of data transfers, the bandwidth allocations for them can be determined similarly. Consequently, the durations of the pull/push gradients of each job will be the same, allowing us to just minimize one of them in the optimization for job scheduling. Then, to facilitate the DML job scheduling to minimize CT-BTI, we model P4INC-AOI as a discrete-time system that operates on time-slots (TS').

Definition 2. *Each* TS *lasts for Δt, being the time granularity for job scheduling and AOI reconfiguration in P4INC-AOI.*

As such, the system time becomes $t \in \{0, \cdots, i \cdot \Delta t, \cdots, T \cdot \Delta t\}$, where i is the index of each TS and T is its maximum. The scheduling scheme of DML jobs and AOI configuration can only be adjusted at the beginning of each TS.

B. MILP Model

We formulate the following MILP model to describe the optimization for scheduling DML jobs in P4INC-AOI.

Notations:

- V: Set of network units in an ODCN based on P4INC-AOI, where each network unit $v \in V$ consists of a PDP switch and a server pool.
- J: Set of DML jobs, each of which contains a PS and a number of workers. For each job $j \in J$, the locations of its PS and workers, its arrival TS, and the volume of data to transmit during pull gradient are all known.
- $N_{\text{INC},v}$: INC capacity of PDP switch $v \in V$, in terms of the number of DML jobs.
- $B_{\text{PDP},v}$: Bandwidth capacity of PDP switch v.
- $P_{\text{oxc},v}$: Number of optical ports on PDP switch v to AOI.
- B_{port}: Bandwidth capacity of each optical port to AOI.
- ε: Time needed for an AOI reconfiguration[2].

[1]Note that, for a practical AOI, two pairs of ports will actually be needed to realize the duplex communication between two PDP switches v and u. But as v and u always communicate in both directions, we can abstract the two port pairs as one and assume that the communication through it is duplex.

[2]As reconfiguring an OXC in AOI interrupts the inter-unit traffic through the affected ports for an unignorable duration, we consider it in the optimization and assume that if AOI needs to be reconfigured by the end of a TS, data will not be transmitted until the reconfiguration has been done in the next TS.

- W_v^j: Number of workers of job $j \in J$ in server pool v.
- L_v^j: Boolean parameter that equals 1 if any worker(s) of job j locate in server pool $v \in V$, and 0 otherwise.
- E_v^j: Boolean parameter that equals 1 if all workers of job j locate in server pool $v \in V$, and 0 otherwise.
- PS_v^j: Boolean parameter that equals 1 if PS of job j locate in server pool v, and 0 otherwise.
- $\tilde{d}_j$: Data volume of each worker for job j, which is generally predictable or can be measured in prior [57].
- a_j: Time for parameter update (in TS') if job j uses its PS in a server pool (when it is not assigned to use INC).
- Δt: Duration of a TS.
- T: Maximum index of TS, *i.e.*, the longest look-ahead time for DML job scheduling.
- $\mathcal{T}$: Set of feasible durations in TS' for job scheduling, *e.g.*, $\mathcal{T} = \{\{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{1,2,3\}\}$ if we have $T = 3$. This set is introduced to ensure that the data transfer of each job runs continuously until finished (it is always assigned to a duration $\tau \in \mathcal{T}$).
- f_τ^i: Boolean parameter that equals 1 if TS i is include in duration τ, and 0 otherwise.
- m/M: A small/large positive constant.

Variables:

- k_v^j: Boolean variable that equals 1 if job j is assigned to use INC in unit v, and 0 otherwise.
- $D_{v,u}^{j,i}$: Volume of data that still needs to be sent between v and u for job j at TS $i \in [0, T]$.
- $B_{v,u}^{j,i}$: Bandwidth assigned to job j between v and u in TS i.
- $\Delta B_{v,u}^{j,i}$: Change on bandwidth allocated to job j between v and u from TS' $i-1$ to i.
- B_j^i: Bandwidth allocated to each worker of job j for pull gradients at TS i. We assume that the all the workers of a job send/receive data to/from their PS at the same data-rate for synchronized training [39], regardless where the workers locate (*i.e.*, in the same network unit or not).
- $P_{v,u}^i$: Number of ports through AOI that are allocated to traffic between v and u in TS i.
- $\Delta P_{v,u}^i$: Change on number of ports through AOI for traffic between v and u from TS' $i-1$ to i.
- t_j: Iteration completion time (ICT) of job j.
- z_j^τ: Boolean variable that equals 1 if job j uses duration τ for data transfer, and 0 otherwise.
- x_j^i: Boolean variable that equals 1 if job j uses TS i for data transfer, and 0 otherwise. ($x_j^i = \sum_{\tau \in \mathcal{T}} z_j^\tau f_\tau^i$).
- $\alpha_{v,u}^i$: Boolean variable that equals 1 if the number of connections through AOI for traffic between v and u increases from TS $i-1$ to TS i, and 0 otherwise.
- $\beta_{v,u}^{j,i}$: Boolean variable that equals 1 if the bandwidth allocated to job j between v and u increases from TS' $i-1$ to TS i, and 0 otherwise.

Objective:

The optimization objective is to minimize the CT-BTI of all the concurrent jobs in J.

$$\text{Minimize} \quad \max\{t_j\}, \forall j \in J. \tag{1}$$

Constraints:

1) INC Resource-related Constraints:

$$\begin{cases} k_v^j \le L_v^j, \\ 0 \le \sum\limits_{v \in V} k_v^j \le 1, \end{cases} \quad \forall j \in J \tag{2}$$

Eq. (2) ensures that global gradient aggregation for job j is performed correctly. Specifically, job j can only use INC resources from the unit where its workers are located, and aggregation must be conducted on a single unit.

$$\sum_{j \in J} k_v^j \cdot x_j^i \le N_{\text{INC},v}, \quad \forall v \in V, \forall i \in T. \tag{3}$$

Eq. (3) ensures that the number of jobs, which are assigned to use INC in each PDP switch at each TS, cannot exceed the INC capacity of the PDP switch.

2) Link Capacity-related Constraints:

$$\begin{cases} B_{v,u}^{j,i} = [k_v^j \cdot (1 - E_v^j) + (1 - k_v^j) \cdot W_v^j] \cdot PS_u^j \cdot B_j^i \cdot x_j^i, \\ B_{v,v}^{j,i} = [k_v^j \cdot E_v^j + (1 - k_v^j \cdot E_v^j) \cdot PS_v^j] \cdot W_v^j \cdot B_j^i \cdot x_j^i, \end{cases} \quad \{v, u : v, u \in V, v \ne u\}, \forall j, i. \tag{4}$$

Eq. (4) determines the correct bandwidth allocation for each job in TS i based on the INC usage and the number and location of its workers and PS.

$$\sum_{j \in J} \left[W_v^j \cdot B_j^i + \sum_u B_{u,v}^{j,i} + (1 - k_v^j) \cdot W_v^j \cdot PS_v^j \cdot B_j^i \right] \cdot x_j^i \le B_{\text{PDP},v}, \quad \{v, u : v, u \in V, v \ne u\}, \forall v, i. \tag{5}$$

Eq. (5) ensures that the total bandwidth allocated to jobs in each unit does not exceed the switch capacity in any TS.

3) AOI Port and Reconfiguration-related Constraints:

$$\begin{cases} P_{v,u}^i = P_{u,v}^i, P_{v,v}^i = 0, \\ \sum\limits_{j \in J} (B_{v,u}^{j,i} + B_{u,v}^{j,i}) \le P_{v,u}^i \cdot B_{\text{port}}, \\ \sum\limits_u P_{v,u}^i \le P_{\text{oxc},v}, \end{cases} \quad \{v, u : v, u \in V, v \ne u\}, \forall i. \tag{6}$$

Eq. (6) enforces that the connections established between units u and v are bidirectional, and that the number of ports used by the connections will not exceed the corresponding port count.

$$\begin{cases} \Delta P_{v,u}^i = P_{v,u}^i - P_{v,u}^{i-1}, \\ \alpha_{v,u}^i \cdot \Delta P_{v,u}^i \ge 0, \\ \Delta P_{v,u}^i + 1 \le M \cdot \alpha_{v,u}^i, \end{cases} \quad \{v, u : v, u \in V, v \ne u\}, \forall i \in [1, T], j. \tag{7}$$

Eq. (7) determines the values of Boolean variables $\{\alpha_{v,u}^i\}$ correctly. Specifically, it sets $\alpha_{v,u}^i = 1$ if an AOI reconfiguration should be triggered in TS i to adapt to the connection increase between units v and u, otherwise, $\alpha_{v,u}^i$ remains as 0.

4) Job Execution-related Constraints:

$$\begin{cases} D_{v,u}^{j,0} = [k_v^j \cdot (1 - E_v^j) + (1 - k_v^j) \cdot W_v^j] \cdot PS_u^j \cdot \tilde{d}_j, \\ D_{v,v}^{j,0} = [k_v^j \cdot E_v^j + (1 - k_v^j \cdot E_v^j) \cdot PS_v^j] \cdot W_v^j \cdot \tilde{d}_j, \end{cases} \quad \{v, u : v, u \in V, v \ne u\}, \forall j. \tag{8}$$

Eq. (8) obtains the initial data volume of job j based on the INC usage and number and location of its workers and PS.

$$\begin{cases} \Delta B_{v,u}^{j,i} = B_{v,u}^{j,i} - B_{v,u}^{j,i-1}, \\ \beta_{v,u}^{j,i} \cdot \Delta B_{v,u}^{j,i} \ge 0, \\ \Delta B_{v,u}^{j,i} \le M \cdot \beta_{v,u}^{j,i}, \end{cases} \quad \{v, u : v, u \in V, v \ne u\}, \forall i \in [1, T], j. \tag{9}$$

Eq. (9) decides the values of Boolean variables $\{\beta_{v,u}^{j,i}\}$, by setting $\beta_{v,u}^{j,i} = 1$ if job j from unit v to unit u requires additional bandwidth in TS i, and 0 otherwise.

$$\begin{cases} D_{v,u}^{j,i+1} = \max\left(0, D_{v,u}^{j,i} - B_{v,u}^{j,i} \cdot \Delta t + \Delta B_{v,u}^{j,i} \cdot \beta_{v,u}^{j,i} \cdot \alpha_{v,u}^{j,i} \cdot \varepsilon\right), \\ D_{v,u}^{j,T} = 0, \end{cases}$$
$$\{v, u : v, u \in V\}, \forall i \in [0, T-1], j. \tag{10}$$

Eq. (10) sets the remaining data volume of each job (the values of variables $\{D_{v,u}^{j,i+1}\}$), where we account for the interruption due to reconfiguration with the last term of the first formula (*i.e.*, a bandwidth penalty of time ε) and use the second one to enforce all the jobs to be completed by the last TS.

$$\begin{cases} \sum\limits_{\tau \in \mathcal{T}} z_j^\tau = 1, \\ t_j = \max\limits_i \left(i \cdot x_j^i\right) \Delta t + a_j \left[\left(1 - \sum\limits_v k_v^j\right) + \sum\limits_v k_v^j \left(1 - \sum\limits_v E_v^j\right)\right], \end{cases}$$
$$\forall j \in J. \tag{11}$$

Eq. (11) ensures that each job is assigned only one duration $\tau \in \mathcal{T}$ for communication. The term $\max\limits_{i \in \tau}\{i \cdot x_j^i\}$ represents the elapsed time since the first TS, while the subsequent term measures the parameter update time. As noted in Section IV-A, the parameter update time is assumed to be 0 if a job uses INC.

Note that, Eqs. (3)-(11) are nonlinear because they contain multiplications of variables. They can be linearized using the standard procedures detailed in Appendix A.

Theorem 1. *The optimization for scheduling DML jobs in P4INC-AOI (described by the MILP above) is $\mathcal{NP}$-hard.*

Proof: We prove the $\mathcal{NP}$-hardness of the optimization by reducing it into the general case of a problem that is known to be $\mathcal{NP}$-hard [58]. Specifically, we first obtain a special case of the optimization by 1) making all the DML jobs arrive at $t = 0$, and 2) forcing $\{N_{\text{INC},v} = 0,\ \forall v \in V\}$ to disable INC on all the PDP switches. Then, we can aggregate the volumes of data to be transferred in the pull gradients step of all the jobs to get a demand matrix $\mathbf{D}_{|V|\times|V|}$, where each element in it indicates the data size of the flow between a pair of network units. Consequently, the optimization for scheduling DML jobs becomes to minimize the longest completion time of flows in $\mathbf{D}_{|V|\times|V|}$ with AOI reconfigurations in the ODCN. This is equivalent to the general case of circuit scheduling for one Coflow demand, and it is known that this problem is $\mathcal{NP}$-hard as long as we have non-zero AOI reconfiguration latency (*i.e.*, $\varepsilon > 0$) [59]. Therefore, we prove the $\mathcal{NP}$-hardness of the original problem, as the reduced problem is $\mathcal{NP}$-hard. ■

V. Algorithm Design for DML Job Scheduling

Since the problem of DML job scheduling is $\mathcal{NP}$-hard, it would become intractable to find its exact solution by solving the MILP formulated in the previous section, when the problem size is relatively large. Hence, in this section, we propose a polynomial-time heuristic to solve it quickly. Specifically, the heuristic decomposes the original scheduling problem into a few smaller subproblems, each of which covers the job scheduling in a small time window of the overall look-ahead time ($[0, T \cdot \Delta t]$), and gradually solves the subproblems.

A. Overall Procedure

Algorithm 1 shows the overall procedure of our proposed heuristic for scheduling DML jobs in P4INC-AOI. *Line* 1 is for the initialization. The for-loop of *Lines* 2-12 handles the job scheduling in iterations, each of which corresponds to a TS. In each iteration, we first update the set of unfinished jobs J and recycle the resources (*i.e.*, bandwidth and INC resources) allocated to completed jobs (*Lines* 3-4). Then, *Line* 5 uses *Algorithm* 2 to obtain the schemes of bandwidth and INC allocation for each unfinished job in J in the current TS, and denotes them with $\{B_{v,u}^{j,i}\}$ and $\{k_v^j\}$. Next, we determine whether an AOI reconfiguration is necessary (*i.e.*, if new connection(s) need to be set up between any pair of network units), invoke the required AOI reconfiguration, and run jobs according to $\{B_{v,u}^{j,i}\}$ and $\{k_v^j\}$ for a TS (*Line* 6). *Lines* 7-11 check each job in J and get its ICT if it is completed. Finally, after all the TS' in the overall look-ahead time have been processed, we return the CT-BTI in *Line* 13.

Algorithm 1: Overall Procedure for Scheduling DML Jobs

```
1  J = ∅;
2  for each i ∈ [0, T] do
3      remove completed jobs from J and reset their resource
       allocation to 0;
4      insert newly-arrived jobs in J;
5      apply Algorithm 2 to get {B^{j,i}_{v,u}} and {k^j_v};
6      reconfigure AOI if necessary and run jobs in J according
       to {B^{j,i}_{v,u}} and {k^j_v} for Δt;
7      for each job j ∈ J do
8          if job j is completed then
9              get t_j based on Equation (11);
10         end
11     end
12 end
13 return t_max = max_{j∈J}(t_j);
```

B. Resource Allocation in each TS

Algorithm 2 describes the procedure of updating the resource allocation scheme of each job in a TS. The for-loop of *Lines* 2-14 allocates INC resources to jobs with more inter-unit traffic to cut down inter-unit traffic, which, in turn, implicitly reduces AOI reconfigurations. This allocation can be effectively solved by leveraging the weighted bipartite matching. Then, *Line* 15 updates the remaining volumes of data transfers in $\{D_{v,u}^{j,i}\}$ according to the flags for INC assignments ($\{k_v^j\}$).

Next, we abstract the PDP switch's physical ports in each network unit as one virtual port for the server pool and another for the AOI. *Line* 16 inputs $\{D_{v,u}^{j,i}\}$ and $\{k_v^j\}$ to *Algorithm* 3 to get the ideal active window in TS' for the usage of each virtual port and sorts them in the descending order of the end time of their windows to prioritize bandwidth allocation to the most time-consuming traffic for minimizing the overall transmission time. The for-loop of *Lines* 17-32 tries to adjust the bandwidth allocation of each port in the sorted order to minimize the CT-BTI of jobs. Specifically, we first put all the jobs that are related to virtual port p in set J_p (*Line* 18), and iteratively increase the bandwidth allocation for jobs in J_p by

a granularity of δB until no more bandwidth can be allocated (*Lines* 17-29). The for-loop of *Lines* 20-28 analyzes each job in J_p to check whether an increase in bandwidth allocation to each of its workers can be accommodated without violating current bandwidth constraints or can be realized by invoking a feasible AOI reconfiguration. If feasible, we record the ICT of the job after applying the related network changes (*Lines* 22-24). Otherwise, we remove the job from J_p (*Line* 26). *Line* 29 finds the job whose projected ICT with bandwidth change is the longest and commits its bandwidth change, and *Lines* 30 updates the network status accordingly. Finally, *Line* 33 returns the resource allocation schemes ($\{B_{v,u}^{j,i}\}$ and $\{k_v^j\}$).

Algorithm 2: Resource Allocation for DML Jobs in a TS

1 get the initial traffic matrix $\tilde{D}_{|V\times J|}$ and create bipartite graph with units V and jobs J with weight $\tilde{d}_{v,j}$;
2 **for** $i = 1$ *to* $\max\left(\tilde{N}_{INC,v}\right)$ **do**
3 run weighted bipartite matching algorithm on $\tilde{D}$;
4 **for** *each matched edge* $(v, j) \in \tilde{D}$ **do**
5 **if** $\tilde{N}_{INC,v} > 0$ **then**
6 assign job j to use INC in unit v;
7 $k_v^j = 1$;
8 remove job j from J;
9 get remaining INC resources in v as $\tilde{N}_{\text{INC},v}$;
10 **else**
11 remove unit v from V ;
12 **end**
13 **end**
14 **end**
15 update $\{D_{v,u}^{j,i}\}$ based on $\{k_v^j\}$;
16 input $\{D_{v,u}^{j,i}\}$ and $\{k_v^j\}$ to *Algorithm* 3 to get the ideal active window for each virtual port and sort them in descending order of their window end time;
17 **for** *each virtual port* p *in sorted order* **do**
18 put jobs in J related to this port to set J_p;
19 **while** $J_p \neq \emptyset$ **do**
20 **for** *each job* $j \in J_p$ **do**
21 increase bandwidth allocation to each worker of job j by δB hypothetically;
22 **if** *the bandwidth change is feasible based on current status of ODCN* **then**
23 get t_j for job j after bandwidth change;
24 record bandwidth change and related AOI reconfiguration for j, if necessary;
25 **else**
26 remove job j from J_p;
27 **end**
28 **end**
29 find the job j^* with the longest ICT t_{j^*} and commit corresponding bandwidth change;
30 update network status to record in $\{B_{v,u}^{j,i}\}$;
31 **end**
32 **end**
33 **return** $\{B_{v,u}^{j,i}\}$ and $\{k_v^j\}$;

C. Estimation of Ideal Active Window for each Virtual Port

Algorithm 3 estimates the ideal active window of each virtual port in the ODCN. The active window of each port p should be determined based on the jobs that will use it to transmit data. However, as the jobs can use INC or not and we need to shorten the longest one of them, their steps of both pull gradients and parameter update need to be considered when allocating bandwidth to them and determining active windows of their ports. For instance, if an INC job shares one port with a non-INC job, we should allocate more bandwidth to the non-INC job to make the two jobs complete simultaneously (*i.e.*, shortening their ICTs) because the parameter update of the non-INC job takes time. Since many factors can affect the ideal active window of each port, it cannot be calculated precisely. Hence, we design *Algorithm* 3 to estimate the lower-bound of the completion time of each window.

Algorithm 3 uses the for-loop of *Lines* 1-16 to estimate the active window of each port. In each iteration, *Lines* 2-4 are for the initialization to get the parameters of a port p, *i.e.*, the total data to transmit, related jobs, and average data transfer time. We then check all the related jobs to find non-INC jobs in them and get the longest parameter update time of them (*Line* 5), and the activate window range of the related jobs is obtained accordingly in *Line* 6. Next, *Lines* 7-14 use binary search to narrow the range according to a preset threshold t_{th} until the capacity of port p is fully utilized to minimize the completion time of its active window. Finally, the ideal active window of port p is obtained in *Line* 15.

Algorithm 3: Getting Ideal Active Window of each Port

1 **for** *each virtual port* p *in ODCN* **do**
2 calculate total data transfer through port p as D_p based on $\{D_{v,u}^{j,i}\}$ and $\{k_v^j\}$;
3 put jobs related to p in set J_p and get total data volume of each job $j \in J_p$ through p to put in $D_{p,j}$;
4 get average data transfer time $\bar{\tau}_p$ on port p based on D_p and the capacity of p;
5 find longest parameter update time $\tau_{\hat{j}}$ of jobs in J_p;
6 set the window range of virtual port p as $[t_0, t_1] = [\bar{\tau}_p, \bar{\tau}_p + \tau_{\hat{j}}]$;
7 **while** $t_1 - t_0 \geq t_{th}$ **do**
8 $\bar{t} = \frac{t_1+t_0}{2}$;
9 **if** $B_{port,p} \geq \sum_{j\in J_P} \frac{D_{p,j}}{t_0-\tau_j}$ **then**
10 $t_1 = \bar{t}$;
11 **else**
12 $t_0 = \bar{t}$;
13 **end**
14 **end**
15 set the ideal time window of port p as from now (the i-th TS) to t_1;
16 **end**
17 **return** ideal time windows of virtual ports in ODCN;

D. Complexity Analysis

The time complexity of our proposed heuristic can be analyzed as follows. As for *Algorithm* 1, its complexity mainly comes from *Algorithm* 2, which consists of three major parts. The first part (*Lines* 1-14) handles the allocation of INC resources, and its complexity is $O(|V| \cdot |J|^2)$. The second part (*Lines* 15-16) uses *Algorithm* 3, which has a complexity of $O(|V| \cdot |J| \cdot \log_2(\frac{\tau_{\hat{j}}}{t_{th}}))$, to get the ideal active window for each virtual port in the ODCN, and thus its complexity is $O(|V| \cdot |J| \cdot \log_2(|\frac{\tau_{\hat{j}}}{t_{th}}|) + |V| \cdot \log_2(|V|))$. The last part (*Lines*

17-32) iteratively allocates bandwidth to the jobs that use each port, and its complexity is $O(|J| \cdot \frac{B}{\delta B} + \frac{|J|^2}{|V|})$. Thus, the overall complexity of *Algorithm* 2 is $O(|V| \cdot |J|^2 + |V| \cdot \log_2(|V|))$.

VI. System Implementation of P4INC-AOI

In this section, we explain the system design and implementation of the data plane of P4INC-AOI.

A. Design Considerations of Data Plane

There are a few approaches in the literature on accelerating DML training with INC in PDP switches (*e.g.*, ATP [39] and GRID [54]). Note that, they do not offload the PS of a DML job completely, but still keep certain part of the PS on a server. Specifically, for a DML job, these existing approaches make PDP switch(es) collect gradients from the workers, aggregate the gradients with INC, and forward the results to the PS, and finally, the PS returns the aggregated gradients to the workers. This design, however, cannot fully explore the mutual benefits of INC and AOI, because the communications between PS' and PDP switches might require connections through AOI, restricting the flexibility of AOI reconfiguration. Hence, we design the INC part of P4INC-AOI by extending the ATP in [39] to fully offload the PS of a DML job to one PDP switch (*i.e.*, the switch directly returns aggregated gradients to workers in each training iteration), thus the communications between PS and PDP switch and between PS and workers are avoided completely, providing P4INC-AOI enhanced flexibility to orchestrate INC and AOI.

In addition to replacing PS completely, we make P4INC-AOI only enable INC for a job if all of its workers locate in the same network unit (*i.e.*, only the single-layer aggregation for local jobs is allowed). The rationale behind this decision is three-fold. First, multi-layer aggregation across units complicates the traffic routing and INC management of each job, and the induced multi-hop routing can make the latencies to workers unequal, restricting the performance of synchronous training. Second, multi-layer aggregation generates traffic that needs to go through AOI and thus is susceptible to AOI reconfiguration. Specifically, AOI reconfiguration can cause packet losses, resulting in switch memory being indefinitely occupied by certain packets with only partial gradients. This can severely degrade INC performance. Third, recent studies have suggested that the majority of DML workloads (*e.g.*, over 99%) use no more than 32 GPUs [60], which is a relatively small number considering that a unit can easily accommodate hundreds of GPUs. This makes it possible that most of the DML workloads can be placed within single units.

We adopt the BytePS framework [10] for each DML job, where the PS in server or PDP switch only handles gradient aggregation and the parameter update is taken care of by the workers. Then, the steps of pull and push gradients can be processed in parallel by leveraging duplex communications.

B. INC for Gradient Aggregation

A Tofino-based PDP switch contains multiple stages, each of which has isolated memory for packet processing. The switch processes packets in the pipeline manner, and thus registers in the same stage cannot be operated on more than once. Therefore, we abstract the switch memory as an array of aggregators. As shown in Fig. 4, aggregator-i uses the i-th registers in all the stages to process a gradient packet. Then, the size of each aggregator is limited by the number of stage in a switch, and it is far less than the size of a gradient generated in each training iteration. Hence, we need to partition each gradient into fragments according to the size of an aggregator, and encapsulate them in multiple packets,

Worker Gradient Push: We design the following fields in the header of a gradient packet: *Service ID*, *Job ID*, *Worker ID*, *Fragment ID*, and *Index*, where *Service ID* indicates whether the packet belongs to an INC job or not and helps a switch determine how to process it accordingly, *Job ID* identifies each DML job, *Worker ID* tells which worker of a job that the packet is for, *Fragment ID* is the index of the gradient fragment in the packet, and *Index* points to the aggregator that should be used to process the packet. We use an independent sliding window to control the packet transmission rate of each worker, which allows to send a new gradient packet after the aggregated result of the previous packet has been received.

As shown in Fig. 4, the aggregators in a switch are arranged in non-overlapping aggregator segments, each of which is labeled with its offset and size (*i.e.*, denoting as <*MemID*, *Size*>). Before starting an INC job, its job scheduler assigns an aggregator segment in one PDP switch to it (*i.e.*, selecting the tuple of <*MemID*, *Size*> for it), according to the instruction from the network controller, to avoid inter-job collisions. Then, in runtime, a worker finds the aggregator for each packet from it by hashing the packet's *Job ID* and *Fragment ID* and combining the result with the *MemID* and *Size* of its job's aggregator segment, and encodes the selected aggregator in *Index* of the packet. Since each packet needs to be hash-indexed to an aggregator, intra-job collisions may occur, which can be easily avoided by each worker with preprocessing (*i.e.*, if a worker detects a collision when assigning an aggregator to a packet, it can redirect the packet to another one). In short, our design lets each worker use deterministic addressing to map packets to aggregators based on combinations of their *Job IDs* and *Fragment IDs*, ensuring that correct aggregation can be realized in each PDP switch. Moreover, this approach also frees up the switches from hash handling, allowing them to process more gradient aggregations with higher throughput.

Switch Gradient Aggregation: When processing each gradient packet, a PDP switch allocates an aggregator to the packet according to its *Index* field and records its *Fragment ID* and *Worker ID*. If the corresponding aggregator is empty, the switch stores the gradient value of the packet and drops it. Otherwise, it compares the *Fragment ID* with that in the aggregator. If the two *Fragment IDs* match, the switch adds the gradient value of the packet to that in the aggregator and increments the worker-count there. Then, if the worker-count equals the number of workers in the job, the aggregator knows that it has summed the gradients from all the workers. It then broadcast the fully aggregated gradient to the workers and reset its registers for subsequent packets. If the worker-count is less than the total number of workers, the aggregator sends

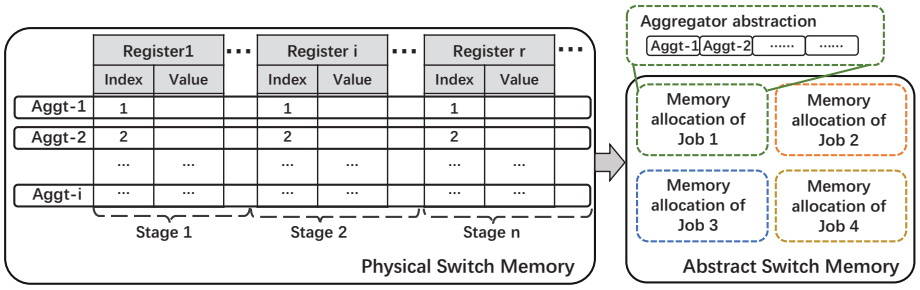


Fig. 4. Data plane design for INC to realize gradient aggregation.

partial aggregation result to the PS. The PS completes the full aggregation and returns the final result, after which the aggregator resets its register. Note that, if the two *Fragment IDs* do not match, an asynchronous error occurs.

Error Control: An asynchronous error happens when packet(s) with a greater *Fragment ID* arrive at a PDP switch while the designated aggregator still has not received all the gradients with the current *Fragment ID*. The switch then flags the error by marking explicit congestion notification (ECN) in a packet, and broadcasts it to all the workers that have sent in packets with incorrect *Fragment IDs*, to let them adjust their gradient transmission rates to prevent asynchronous arrivals of gradient fragments in the future. In this work, we make each worker use the additive increase multiplicative decrease (AIMD) algorithm [39] to adjust its sliding window size. The AIMD algorithm works as follows. At the beginning, each worker maintains the same initial window size. Then, when it receives a packet for aggregated gradient, it increases its window size by one maximum transmission unit (MTU) until reaching a threshold, which was set according to the aggregators and bandwidth allocated to the worker's job and the round-trip time (RTT) between the workers and the switch. Upon receiving an ECN-marked packet, the worker reduces its window size by half and updates the threshold accordingly.

In addition to asynchronous errors, packet losses can happen and cause errors, which can make aggregators wait indefinitely. To address this issue, we implement a retransmission mechanism in workers. If a worker receives an aggregated gradient packet whose *Fragment ID* is greater than what the worker expects, it retransmits the previous gradient packet. Upon receiving such a duplicated gradient packet, the switch will resend the corresponding aggregated gradient to the worker. Note that, in order to realize the retransmission mechanism, the switch needs to retain a copy of the aggregated gradient for each aggregator. This can be accomplished by leveraging the shadow copy in [38], which allows a switch to retransmit a lost aggregated gradient packet even if it has already started reusing the related aggregator for the next gradient fragment.

VII. Performance Evaluation

In this section, we first give the metrics and benchmarks for performance comparison, and then build a small-scale testbed with Tofino-based PDP switches [51] and an OXC to evaluate P4INC-AOI for DML training, and finally perform simulations to test the performance of P4INC-AOI in larger scales.

A. Performance Metrics and Benchmarks

Metrics: We adopt the following metrics for performance evaluations: 1) the number of AOI reconfigurations, 2) total job completion time (JCT) (small-scale experiments), and 3) the CT-BTI (large-scale simulations). The metrics are chosen to test the mutual benefits of INC and AOI for DML training. Specifically, the first metric is used to check how the malleable traffic matrix shaped by INC can reduce the frequency of AOI reconfigurations, and the second and third metrics measure the performance of DML training in P4INC-AOI.

Benchmarks: We consider four scenarios to illustrate the performance of P4INC-AOI and our proposed job scheduling algorithms. In the system aspect, we adopt both P4INC-AOI and a traditional ODCN without INC, where the traditional ODCN uses the same hardware configuration as P4INC-AOI, except for that we do not enable INC in it. In the algorithm aspect, we consider a straightforward first-come-first-served scheduling algorithm (FCFS) and our time-window-based scheduling algorithm (TW). The FCFS provisions DML jobs one by one in the sequence of their arrivals and allocates the multi-dimensional resources in ODCN (*i.e.*, IT resources in server pools, memory in PDP switches (if INC is enabled), and bandwidth on network links) to each job to shorten their JCT/CT-BTI greedily. Then, we combine the system and algorithm aspects to obtain the following four scenarios.

- **noINC-FCFS:** The traditional ODCN without INC uses FCFS to schedule DML jobs.
- **noINC-TW:** The traditional ODCN without INC uses our TW to schedule jobs.
- **INC-FCFS:** P4INC-AOI uses FCFS to schedule jobs.
- **INC-TW:** P4INC-AOI uses TW to schedule jobs.

B. Experimental Evaluation

Settings: We build a small-scale but realistic ODCN testbed with off-the-shelf components. It consists of four racks, *i.e.*, two GPU racks and two CPU racks, respectively. Each GPU rack contains 4 servers, each of which is equipped with one 1080Ti GPU, one 40-Gbps Mellanox ConnectX-5 network interface card (NIC), and two Intel 32-core CPUs at 2.2 GHz. The software environment of each server is Ubuntu 18.04 and CUDA 10.2 with Mellanox OFED 4.9-6.0.6.0 NIC driver, and it runs PyTorch to execute the workers of DML jobs. Each worker utilizes the distributed data parallel module provided by PyTorch. The bandwidth increment granularity for workers (*i.e.*, δB in *Algorithm* 2) was set as 10 Gbps.

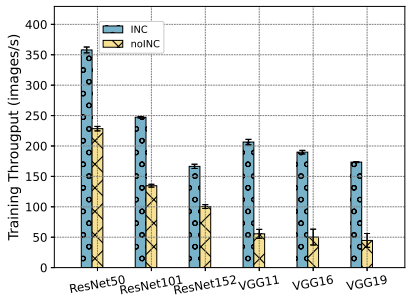


Fig. 5. Results on training throughput of DML models.

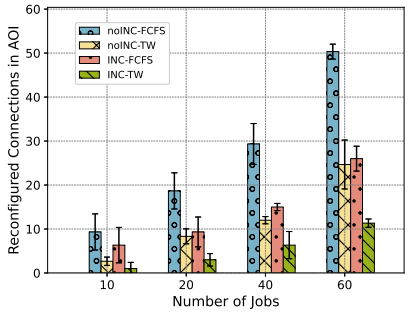
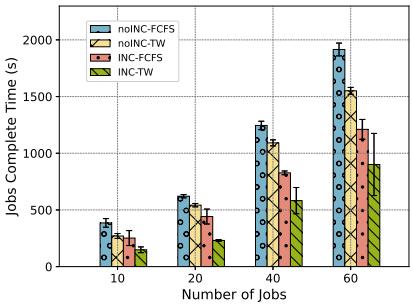
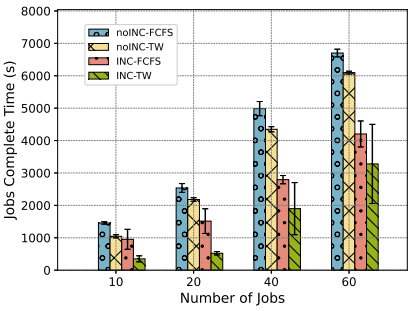


Fig. 6. Results on AOI reconfigurations.



(a) ResNet50



(b) VGG16 model

Fig. 7. Results of concurrent jobs in Poisson process.

The ToR switch of each rack is a 32×40-Gbps Tofino-based PDP switch with software development environment (SDE) 9.7.0. The AOI is based on a 32×32 commercial OXC, and each of its port connects to an optical port on a ToR switch. In the control plane, the network controller can set up and reconfigure connections in the OXC by leveraging the OpenFlow protocol, and install INC and routing policies in PDP switches through Barefoot runtime interface via the secure shell (SSH) protocol, including sending *Job IDs* to distinguish INC and non-INC jobs. INC is implemented with P4-16 runtime programs that are executed in PDP switches using the Tofino native architecture (TNA).

As for the DML workloads, we consider a few well-known machine learning models: ResNet50, ResNet101, ResNet152, VGG11, VGG16, and VGG19. We first conduct experiments to measure the performance gain achieved by INC. The DML job of each model consists of two workers and a PS in BytePS [10], and trains with the standard 3-channel and 224×224 pixel images generated with the function *torch.rand()*. The batch size is set as 32 for all the models. We fix the duration of a TS as 1 second, and obtain the training throughput of each model after a warm-up. Fig. 5 shows the experimental results on DML training throughput, indicating that INC achieves $1.56\times$, $1.83\times$, $1.65\times$, $3.7\times$, $3.7\times$, and $3.8\times$ throughput improvements over the noINC cases for ResNet50, ResNet101, ResNet152, VGG11, VGG16, and VGG19, respectively. In general, the performance gains are larger for bandwidth-intensive workloads (VGG) than for computing-intensive workloads (ResNet). In the following experiments, we choose VGG16 (with 528 MB model size) and ResNet50 (with 102 MB model size) to represent bandwidth-intensive and computing-intensive DML workloads, respectively.

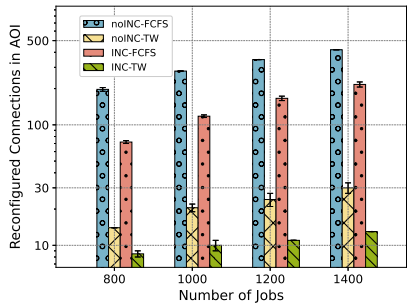
Next, we evaluate the AOI reconfigurations and total JCT of the 4 scenarios outlined in Section VII-A, with the number of concurrent DML jobs in each experiment chosen from $\{10, 20, 40, 60\}$. Jobs come in according to the Poisson process with an arrival rate of one job per TS. This time, we make each job include $\{2, 4\}$ workers randomly and one PS. For each job, it can wait in the scheduling queue before being served, and thus the JCT of a job includes its waiting time and training time. In each experiment, we average the results obtained in three independent runs to get each data point.

AOI reconfigurations: We record the total number of reconfigured connections in each experiment to reflect the complexity of AOI reconfigurations. The results are shown in Fig. 6, which indicate that INC-TW always reconfigures the smallest number of AOI connections followed by noINC-TW. As for the scenarios with FCFS, the one with INC (INC-FCFS) reconfigures less connections than that without. These results confirm both the effectiveness of our proposed job scheduling algorithm and the benefit of INC. Specifically, INC can reduce the number of reconfigured connections by 48.8% on average, and TW reconfigures 47.5% less AOI connections over FCFS. This suggests that P4INC-AOI with TW significantly reduces AOI reconfigurations and mitigates the mismatch between traffic generated by DML jobs and topology of AOI.

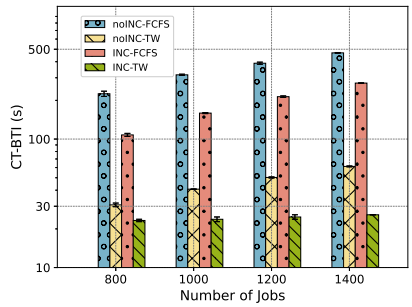
Total JCT: We then measure the total JCTs of DML jobs for ResNet50 and VGG16, and plot the results in Fig. 7. When FCFS is used, INC reduces total JCT by 33.57% and 43.96% on average for ResNet50 and VGG16, respectively (comparing INC-FCFS to noINC-FCFS). Our proposed job scheduling algorithm reduces total JCT by 46.66% and 56.34% on average for ResNet50 and VGG16, respectively (comparing INC-TW to noINC-TW). The results also reveal that TW achieves

TABLE I
RESULT OF SMALL-SCALE SIMULATIONS

	MILP			INC-FCFS				INC-TW				INC-TW-NoRecon			
Jobs	Time (s)	CT-BTI (s)	AOI	Time (s)	CT-BTI (s)	$\delta(\%)$	AOI	Time (s)	CT-BTI (s)	$\delta(\%)$	AOI	Time (s)	CT-BTI (s)	$\delta(\%)$	AOI
40	84.6	3.4	1	0.6	6.0	43.3	1	0.8	4.1	17.1	1	0.7	5.1	33.3	0
50	105.3	6.9	1	0.8	11.2	38.4	1	1.0	8.2	15.9	1	0.9	9.8	29.6	0
60	357.5	14.8	1	0.9	23.8	37.8	2	1.1	17.7	16.4	2	1.0	21.7	31.8	0
80	–	–	–	1.3	29.9	–	5	1.5	19.8	–	4	1.4	21.7	–	0
100	–	–	–	1.6	31.8	–	11	2.1	26.7	–	5	1.7	30.7	–	0



(a) AOI reconfigurations



(b) CT-BTI

Fig. 8. Results of large-scale simulations (different numbers of DML jobs).

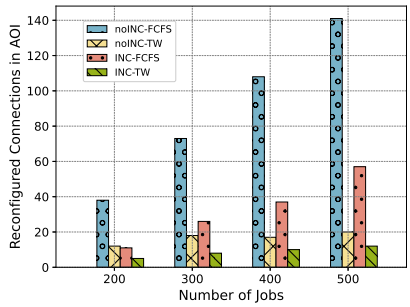
an average reduction of 12.34% in total JCT for ResNet50 and 12.7% for VGG16 in scenarios without INC (noINC-TW *versus* noINC-FCFS), and an average reduction of 29.62% for ResNet50 and 32.0% for VGG16 in scenarios with INC (INC-TW *versus* INC-FCFS). Therefore, both computing-intensive (ResNet50) and bandwidth-intensive (VGG16) jobs benefit from our proposal of P4INC-AOI using TW.

C. Simulation Evaluation

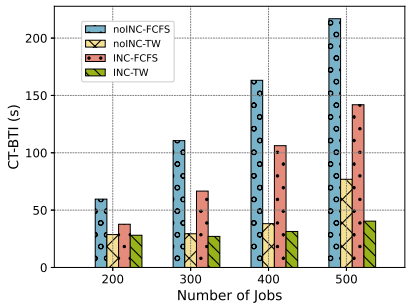
As experiments in the small-scale ODCN testbed cannot verify the performance of our proposal for large-scale problems, we use numerical simulations to further evaluate it. The simulations first compare the results from our heuristic with the exact ones from the MILP formulated in Section IV-B, to check how the heuristic can approximate the exact solutions for small-scale problems, and then evaluate the four heuristic scenarios mentioned in the previous subsection.

Settings: The MILP is solved with the Gurobi solver, while we implement the heuristic with Python, and both of them run on a computer equipped with an Intel Core i5-10500 CPU and 16 GB memory. As for the large-scale simulations, we consider an ODCN with 64 units at most, each of which contains 64 servers and a PDP aggregation switch that can accommodate 4 jobs with INC at most. Each unit connects to the AOI through 24 40-Gbps optical ports. To verify that our proposed algorithms can work for multi-application scenarios in ODCNs, we consider two types of jobs: DML jobs and MapReduce jobs, both arriving according to a Poisson process with an average arrival rate of 20 jobs per TS, where TS is set to 1 second. For each DML job, the number of workers is randomly selected from $\{2, 4, 8, 16\}$ based on the GPU request sizes of DML jobs in Philly's trace [60]. Each worker uniformly chooses its data size of one push step within $[102, 548]$ MB, emulating the parameter sizes of DML models such as ResNet50, ResNet101, ResNet152, VGG11, VGG16, and VGG19. The MapReduce jobs are generated using the Facebook's trace [61], which contains 500 MapReduce jobs and their numbers of workers and data volumes. Same as that in the experimental setup, we set the bandwidth increment step for workers as 10 Gbps. Note that, the locations of workers in each job are the inputs to our scheduling problem, and thus for fair comparisons, we randomly place workers in each simulation but try to place as many workers of each job in one unit as possible, to maximize traffic localization.

Benchmarking with MILP: The results of small-scale simulations with 8 units are shown in Table I. MILP always provides the minimum CT-BTI and the smallest number of AOI reconfigurations, but it is time-consuming, taking over 5 minutes to solve the scheduling for 60 jobs. The running time of INC-FCFS and INC-TW is orders of magnitudes shorter than that of MILP. INC-FCFS runs the fastest but its gaps to the optimal solutions from MILP are larger than those of INC-TW. The gaps on CT-BTI of INC-FCFS are within $[37.8\%, 43.3\%]$. INC-TW outperforms INC-FCFS by reducing the CT-BTI deviation from the optimal solution to within $[15.9\%, 17.1\%]$ and invoking fewer AOI reconfigurations. This is because INC-TW prioritizes INC resources to jobs producing heavier inter-unit traffic to effectively ease contention on AOI connections and accelerates DML training by allocating more bandwidth to jobs with longer ICT. Moreover, to check the effect of AOI reconfiguration on DML training, we consider the approach that uses the procedure of INC-TW but does not allow AOI reconfiguration (*i.e.*, INC-TW-NoRecon in Table I), and as no reconfiguration is allowed, we pre-configure the AOI with uniform connectivity between network unit pairs. INC-TW consistently outperforms INC-TW-NoRecon

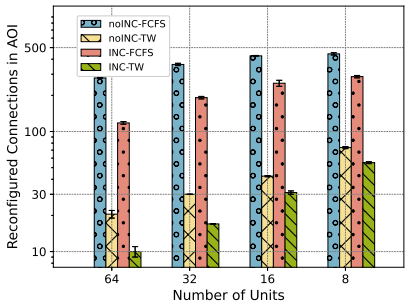


(a) AOI reconfigurations

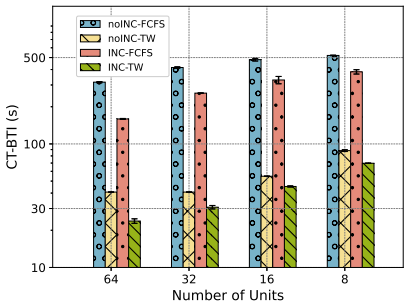


(b) CT-BTI

Fig. 9. Large-scale simulations for different numbers of MapReduce jobs.



(a) AOI reconfigurations



(b) CT-BTI

Fig. 10. Results of large-scale simulations (different numbers of units).

in CT-BTI, and performance gap is within $[9.6\%, 24.4\%]$. This relatively significant gap justifies the necessity of AOI reconfiguration even in small-scale ODCNs.

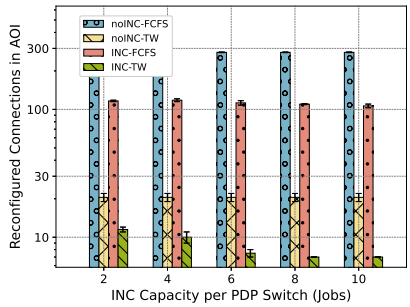
Performance with different numbers of jobs: Fig. 8 plots the results on AOI reconfigurations and CT-BTI when the number of DML jobs increases from 800 to $1,400$. INC-TW still consistently reconfigures the fewest number of AOI connections and delivers the shortest CT-BTI, aligning with the experimental results in Figs. 6-7. As for all the scenarios, INC still effectively reduces the number of AOI configurations over those without it. Specifically, the average reductions on AOI reconfigurations are 55.4% for INC-FCFS over noINC-FCFS and 50.3% for INC-TW over noINC-TW. Fig. 6 also verify the benefit of our TW algorithm, *i.e.*, it respectively achieves 92.8% and 91.7% reductions on AOI reconfigurations for noINC and INC scenarios. In all, the results on AOI reconfigurations confirm the advantage of symbiosis of INC and our TW algorithm. Meanwhile, it is interesting to notice that different from the JCT results in Fig. 7, the CT-BTI of INC-FCFS is longer than that of noINC-TW. This suggests that for large-scale problems, the benefit of our TW algorithm on reducing JCT becomes more significant. Our TW algorithm reduces the CT-BTI by on average 86.8% and 85.6% in the noINC and INC scenarios, respectively.

Fig. 9 shows the results of AOI reconfigurations and CT-BTI as functions of the number of MapReduce jobs. Similar to the results for DML jobs, INC-TW consistently achieves the fewest AOI reconfigurations and the shortest CT-BTI, demonstrating the effectiveness of our TW algorithm in generalizing tasks involving collective communications. On average, INC-TW respectively reduces the number of AOI reconfigurations and CT-BTI by 89.5% and 72.6% over noINC-FCFS.

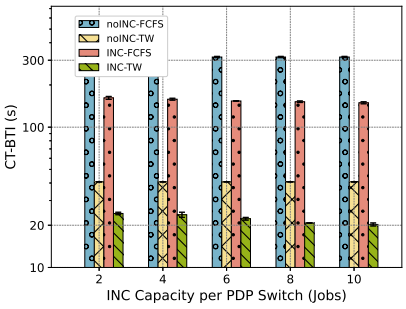
Performance with different ODCN scales: Fig. 10 shows the results for ODCNs with different numbers of units when the number of DML jobs is fixed as $1,000$. We find that as the scale of the ODCN increases, both the AOI reconfigurations and the CT-BTI decrease for each scheme. This is because a larger-scale ODCN has more INC and bandwidth resources, making DML scheduling easier. noINC-TW consistently outperforms INC-FCFS. Once again, this suggests that the benefits of our TW algorithm on reducing AOI reconfigurations and CT-BTI become more significant as the problem scale increases. Overall, INC-TW achieves the lowest AOI reconfigurations compared to other three approaches.

Performance with different INC capacities: Fig. 11 illustrates the impact of INC capacity of PDP switches on the performance of P4INC-AOI. As for the scenarios without INC, it is easy to understand that changing INC capacity will not affect the results on AOI reconfigurations or CT-BTI. For INC-FCFS and INC-TW, both AOI reconfigurations and CT-BTI decrease with INC capacity, as larger INC capacity eases the scheduling of DML jobs. The performance gain almost diminishes as we further increase INC capacity (*i.e.*, beyond 8) because bandwidth bottlenecks turn to the major constraint.

Performance with different TS lengths: Finally, we explore the impact of the length of TS on the performance of DML job scheduling, and Fig. 12 shows the results. In Fig. 12(a), AOI reconfigurations first decrease and then converge as TS length increases. However, a longer TS length results in a longer CT-BTI across all the four scenarios, due to the coarser job scheduling that can reduce resource utilization in TS'. Meanwhile, for noINC-TW and INC-TW, the increases on CT-BTI are much smaller than those of noINC-FCFS and INC-FCFS. Hence, our TW algorithm provides a better opportunity

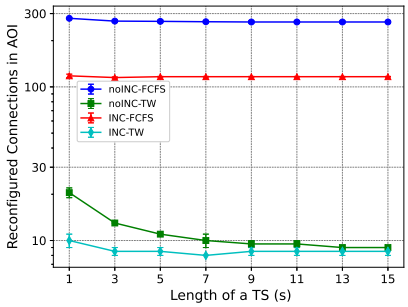


(a) AOI reconfigurations

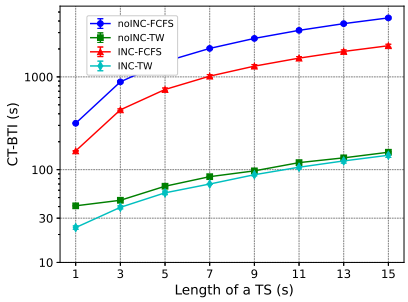


(b) CT-BTI

Fig. 11. Results of large-scale simulations (different INC capacities).



(a) AOI reconfigurations



(b) CT-BTI

Fig. 12. Results of large-scale simulations (different TS lengths).

to properly balance the tradeoff between AOI reconfigurations and the CT-BTI by changing TS length.

VIII. Conclusion

In this paper, we proposed P4INC-AOI, which is an ODCN framework empowered by INC using the off-the-shelf PDP switches based on Tofino ASICs. P4INC-AOI jointly leverages innovations in the control and data planes to accelerate training of DML jobs as well as to reshape/absorb inter-rack/pod traffic for minimizing AOI reconfigurations in an ODCN. We first proposed the algorithms (*i.e.*, an MILP model and a time-efficient heuristic), with which the control plane can effectively orchestrate INC and AOI to schedule DML jobs efficiently. Then, we designed a gradient aggregation principle to mitigate the synchronization errors caused by traffic congestion or switch resource conflicts in the data plane. The performance of P4INC-AOI was validated with both small-scale experiments and large-scale simulations. The results showed that P4INC-AOI achieved significant reductions in JCT and AOI reconfigurations when scheduling concurrent DML jobs.

Acknowledgments

This work was supported by the NSFC project 62371432.

Appendix A
Linearization Method

In this appendix, we describe the methods used to linearize the nonlinear constraints in the MILP (*i.e.*, Eqs. (3)-(11)).

The **multiplication of Boolean variables** a and b, denoted by $z = a \cdot b$, can be linearized using the following constraints.

$$\begin{cases} z \le a, z \le b, \\ z \ge a + b - 1. \end{cases} \tag{12}$$

The **multiplication of Boolean variable** a **and real variable** b, denoted by $z = a \cdot b$, can be linearized using Eq. (13), where M is a sufficiently large positive constant.

$$\begin{cases} -aM \le z \le aM, \\ b - (1-a)M \le z \le b + (1-a)M. \end{cases} \tag{13}$$

The **Max operation** $z = \max(a, b)$, where a and b are real variables, can be linearized by introducing two auxiliary variables u and v and imposing the following constraints.

$$\begin{cases} z \ge a, z \ge b, \\ z \le a + M(1-u), z \le b + M(1-v), \\ u + v \ge 1. \end{cases} \tag{14}$$

Similarly, the **Min operation** $z = \min(a, b)$ can be replaced by the following linear equations.

$$\begin{cases} z \le a, z \le b, \\ z \ge a + M(1-u), z \ge b + M(1-v), \\ u + v \ge 1. \end{cases} \tag{15}$$

The above methods can be concatenated or nested to cope with more complex scenarios, *e.g.*, linearizing $z = b \prod_i a_i$.

References

[1] Y. Gong *et al.*, "A survey on dataset quality in machine learning," *Inf. Softw. Technol.*, vol. 162, p. 107268, Oct. 2023.

[2] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[3] K. Cooper, "OpenAI GPT-3: Everything you need to know [Updated]," Sept. 2023. [Online]. Available: https://www.springboard.com/blog/data-science/machine-learning-gpt-3-open-ai/.

[4] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. of NeurIPS 2012*, pp. 1223–1231, Dec. 2012.

[5] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. of OSDI 2016*, pp. 265–283, Nov. 2016.

[6] P. Moritz *et al.*, "Ray: a distributed framework for emerging AI applications," in *Proc. of OSDI 2018*, pp. 561–577, Oct. 2018.

[7] Z. Tang *et al.*, "Communication-efficient distributed deep learning: A comprehensive survey," *arXiv preprint arXiv:2003.06307*, Mar. 2020. [Online]. Available: https://arxiv.org/abs/2003.06307.

[8] H. Zhang *et al.*, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *Proc. of USENIX ATC 2017*, pp. 181–193, Jun. 2017.

[9] L. Luo *et al.*, "Parameter hub: a rack-scale parameter server for distributed deep neural network training," in *Proc. of ACM SoCC 2018*, pp. 41–54, Oct. 2018.

[10] Y. Jiang *et al.*, "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters," in *Proc. of OSDI 2020*, pp. 463–479, Nov. 2020.

[11] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. of ACM SIGCOMM 2008*, pp. 1–12, Aug. 2008.

[12] P. Lu *et al.*, "Highly-efficient data migration and backup for Big Data applications in elastic optical inter-data-center networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[13] P. Lu and Z. Zhu, "Data-oriented task scheduling in fixed- and flexible-grid multilayer inter-DC optical networks: A comparison study," *J. Lightw. Technol.*, vol. 35, pp. 5335–5346, Dec. 2017.

[14] W. Lu *et al.*, "AI-assisted knowledge-defined network orchestration for energy-efficient data center networks," *IEEE Commun. Mag.*, vol. 58, pp. 86–92, Jan. 2020.

[15] M. Zhang *et al.*, "Gemini: Practical reconfigurable datacenter networks with topology and traffic engineering," *arXiv preprint arXiv:2110.08374*, Oct. 2021. [Online]. Available: https://arxiv.org/abs/2110.08374.

[16] A. Greenberg *et al.*, "VL2: a scalable and flexible data center network," in *Prof. of ACM SIGCOMM 2009*, pp. 51–62, Aug. 2009.

[17] B. Niu *et al.*, "Visualize your IP-over-optical network in realtime: A P4-based flexible multilayer in-band network telemetry (ML-INT) system," *IEEE Access*, vol. 7, pp. 82 413–82 423, 2019.

[18] N. Jouppi *et al.*, "TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proc. of ISCA 2023*, pp. 1–14, Jun. 2023.

[19] N. Farrington *et al.*, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proc. of ACM SIGCOMM 2010*, pp. 339–350, Aug. 2010.

[20] K. Chen *et al.*, "OSA: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Trans. Netw.*, vol. 22, pp. 498–511, Mar. 2013.

[21] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

[22] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[23] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[24] W. Lu, Z. Zhu, and B. Mukherjee, "On hybrid IR and AR service provisioning in elastic optical networks," *J. Lightw. Technol.*, vol. 33, pp. 4659–4669, Nov. 2015.

[25] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[26] J. Zerwas, W. Kellerer, and A. Blenk, "What you need to know about optical circuit reconfigurations in datacenter networks," in *Proc. of ITC 2021*, pp. 1–9, Aug./Sept. 2021.

[27] H. Liu *et al.*, "Lightwave Fabrics: At-scale optical circuit switching for datacenter and machine learning systems," in *Proc. of ACM SIGCOMM 2023*, pp. 499–515, Aug. 2023.

[28] G. Porter *et al.*, "Integrating microsecond circuit switching into the data center," in *Proc. of ACM SIGCOMM 2013*, pp. 447–458, Aug. 2013.

[29] H. Ballani *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proc. of ACM SIGCOMM 2020*, pp. 782–797, Jul. 2020.

[30] X. Xiao *et al.*, "Silicon photonic Flex-LIONS for bandwidth-reconfigurable optical interconnects," *J. Sel. Top. Quantum Electron.*, vol. 26, p. 3700210, Mar./Apr. 2020.

[31] L. Poutievski *et al.*, "Jupiter evolving: Transforming Google's datacenter network via optical circuit switches and software-defined networking," in *Proc. of ACM SIGCOMM 2022*, pp. 66–85, Aug. 2022.

[32] S. Zhao, P. Cao, and X. Wang, "Understanding the performance guarantee of physical topology design for optical circuit switched data centers," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, pp. 1–24, Mar. 2021.

[33] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[34] N. Zilberman, "In-network computing," Apr. 2019. [Online]. Available: https://www.sigarch.org/in-network-computing-draft/.

[35] X. Xie, H. Yang, and Z. Zhu, "P4INC-AOI: When in-network computing meets all-optical interconnect for adaptive and low-latency optical DCN," in *Proc. of OFC 2023*, pp. 1–3, Mar. 2023.

[36] L. Liu *et al.*, "Online job scheduling for distributed machine learning in optical circuit switch networks," *Knowl-Based Syst.*, vol. 201-202, pp. 106 002–106 015, Aug. 2020.

[37] H. Wang, Z. Liu, and H. Shen, "Job scheduling for large-scale machine learning clusters," in *Proc. of CoNEXT 2020*, pp. 108–120, Nov. 2020.

[38] A. Sapio *et al.*, "Scaling distributed machine learning with in-network aggregation," in *Proc. of NSDI 2021*, pp. 785–808, Apr. 2021.

[39] C. Lao *et al.*, "ATP: In-network aggregation for multi-tenant learning," in *Proc. of NSDI 2021*, pp. 741–761, Apr. 2021.

[40] M. Zinkevich, M. Weimer, L. Li, and A. Smola, "Parallelized stochastic gradient descent," in *Proc. of NeurIPS 2010*, pp. 1–9, Dec. 2010.

[41] S. Shalev and S. Ben, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University, 2014.

[42] J. Verbraeken *et al.*, "A survey on distributed machine learning," *arXiv preprint arXiv:1912.09789*, Dec. 2019. [Online]. Available: https://arxiv.org/abs/1912.09789.

[43] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *J. Parallel Distrib. Comput.*, vol. 69, pp. 117–124, Feb. 2009.

[44] J. Verbraeken *et al.*, "A survey on distributed machine learning," *CSUR*, vol. 53, pp. 1–33, Mar. 2020.

[45] Z. Chen *et al.*, "Rina: Enhancing ring-allreduce with in-network aggregation in distributed model training," *arXiv preprint arXiv:2407.19721*, pp. 1–12, Jul. 2024. [Online]. Available: https://arxiv.org/abs/2407.19721.

[46] M. Teh, S. Zhao, P. Cao, and K. Bergman, "Enabling quasi-static reconfigurable networks with robust topology engineering," *IEEE/ACM Trans. Netw.*, vol. 31, pp. 1056–1070, Jun. 2023.

[47] P. Cao *et al.*, "Threshold-based routing-topology co-design for optical data center," *IEEE/ACM Trans. Netw.*, vol. 31, pp. 2870–2885, Mar. 2023.

[48] G. Wang *et al.*, "Domino: Eliminating communication in LLM training via generic tensor slicing and overlapping," *arXiv preprint arXiv:2409.15241*, pp. 1–16, Sept. 2024. [Online]. Available: https://arxiv.org/abs/2409.15241.

[49] NetFPGA. [Online]. Available: https://netfpga.org/.

[50] J. Min *et al.*, "Gimbal: Enabling multi-tenant storage disaggregation on SmartNIC JBOFs," in *Proc. of ACM SIGCOMM 2021*, pp. 2106–122, Aug. 2021.

[51] Intel Tofino. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.

[52] R. Segal, C. Avin, and G. Scalosub, "SOAR: Minimizing network utilization with bounded in-network computing," in *Proc. of CoNEXT 2021*, pp. 16–29, Dec. 2021.

[53] N. Gebara, M. Ghobadi, and P. Costa, "In-network aggregation for shared machine learning clusters," in *Proc. of MLSys 2021*, pp. 829–844, Mar. 2021.

[54] J. Fang *et al.*, "GRID: Gradient routing with in-network aggregation for distributed training," *IEEE Trans. Netw. Serv. Manag.*, vol. 31, pp. 2267–2280, Oct. 2023.

[55] Intel Intelligent Fabric Processors. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html.

[56] E. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, pp. 87 094–87 155, Jun. 2021.

[57] Q. Weng *et al.*, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *Proc. of NSDI 2022*, pp. 945–960, Apr. 2022.

[58] M. Garey and D. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman and Co. New York, 1979.

[59] X. Huang and Sun, "Sunflow: Efficient optical circuit scheduling for Coflows," in *Proc. of CoNEXT 2016*, pp. 297–311, Dec. 2016.

[60] M. Jeon *et al.*, "Analysis of large-scale multi-tenant gpu clusters for dnn training workloads," in *Proc. of USENIX ATC 2019*, pp. 947–960, Jun. 2019.
[61] Coflow-Benchmark. [Online]. Available: https://github.com/coflow/coflow-benchmark.