RESEARCH-ARTICLE

A2TP: Aggregator-aware In-network Aggregation for Multi-tenant Learning

ZHAOYI LI, Central South University, Changsha, Hunan, China

JIAWEI HUANG, Central South University, Changsha, Hunan, China

YIJUN LI, Central South University, Changsha, Hunan, China

AIKUN XU, Central South University, Changsha, Hunan, China

SHENGWEN ZHOU, Central South University, Changsha, Hunan, China

JINGLING LIU, Central South University, Changsha, Hunan, China

View all

Published: 08 May 2023

EuroSys '23: Eighteenth European Conference on Computer Systems
May 8 - 12, 2023
Rome, Italy

Conference Sponsors:
SIGOPS

A^{2}TP: Aggregator-aware In-network Aggregation for Multi-tenant Learning

Zhaoyi Li, Jiawei Huang, Yijun Li, Aikun Xu, Shengwen Zhou, Jingling Liu, Jianxin Wang
Central South University
Changsha, Hunan, China
{lizhaoyi,jiaweihuang,yijunli,aikunxu,zhousw,jinglingliu}@csu.edu.cn,jxwang@mail.csu.edu.cn

Abstract

Distributed Machine Learning (DML) techniques are widely used to accelerate the training of large-scale machine learning models. However, during training iterations, gradients need to be frequently aggregated across multiple workers, resulting in communication bottleneck. To reduce the communication overhead of DML, several In-Network Aggregation (INA) protocols are proposed to reduce the volume of aggregation traffic by offloading aggregation functions into switches, thus alleviating network bottlenecks. Nevertheless, these protocols couple the congestion control of in-switch aggregator resources and link bandwidth resources, together with the straggler-oblivious manner in aggregator allocation, leading to low aggregation efficiency.

To solve the above problem, we propose an Aggregator-aware in-network Aggregation Transmission Protocol (A^{2}TP), which adopts two congestion windows to decouple the congestion control of two resources and combines with the straggling estimation scheme to efficiently allocate aggregator resources according to the straggler degree for multiple jobs, eliminating the impact of straggler jobs on the overall aggregation process. We implement A^{2}TP at P4-programmable switch and a kernel bypass protocol stack at the end-host. The evaluation results show that A^{2}TP reduces the training time by up to 66% than the state-of-the-art INA protocols in real-world benchmark models.

CCS Concepts: • **Computing methodologies → Machine learning**; • **Computer systems organization → Distributed architectures**; • **Networks → Network protocols**.

Keywords: in-network aggregation, machine learning, transmission protocol

EuroSys '23, May 8–12, 2023, Rome, Italy

ACM ISBN 978-1-4503-9487-1/23/05…$15.00
https://doi.org/10.1145/3552326.3587436

ACM Reference Format:
Zhaoyi Li, Jiawei Huang, Yijun Li, Aikun Xu, Shengwen Zhou, Jingling Liu, Jianxin Wang. 2023. A^{2}TP: Aggregator-aware In-network Aggregation for Multi-tenant Learning. In *Eighteenth European Conference on Computer Systems (EuroSys '23), May 8–12, 2023, Rome, Italy.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3552326.3587436

1 Introduction

In recent years, machine learning (ML) is emerging as an indispensable component in a wide range of enterprise applications, such as natural language processing, data mining and computer vision. With the growing scale of training dataset and model size (e.g., Switch Transformer [1] has 1.6 trillion parameters), the distributed machine learning (DML) frameworks [2, 3] are widely adopted in large-scale clusters with hundreds of nodes and GPUs to reduce training time. A common DML technique is data-parallel training, which partitions the dataset into multiple workers and parallelly runs training jobs.

However, during the phases of synchronous model updates among workers, network communication has become the bottleneck to slow down the training process. Recently, a series of in-network aggregation (INA) schemes such as SwitchML [4], PANAMA [5] and ATP [6] are proposed to alleviate the communication bottleneck for DML. The basic mechanism is that the workers stream the gradients to switch, and utilize a block of switch memory (organized as aggregators) to store and aggregate the gradients. After the gradient aggregation completes, the switch broadcasts the aggregated results back to each worker. Through the flexible use of switch aggregators, these protocols significantly reduce the aggregation traffic volume, therefore mitigating the network bottlenecks and accelerating the training process.

Unfortunately, the size of switch memory is very limited (e.g. ~10MB on Intel Tofino switch [7]), while there are massive concurrent jobs in a large-scale cluster [8], making the memory resource insufficient. For example, if SwitchML is enabled, it requires approximately 1MB of switch memory under 100Gbps link bandwidth to support line-rate aggregation for each training job [4]. Moreover, several optional functions such as network traffic measurement [9] and packet scheduling [10] are also deployed on the programmable switch, further compressing the available resources for in-network

aggregation. Due to high requirement from training jobs, it becomes challenging to make effective use of limited switch memory for INA schemes.

The existing INA schemes employ end-to-end transmission protocols to guarantee the correctness of aggregation. However, these solutions present many fundamental issues. (1) The end-to-end transmission protocols regard the network as a black box. SwitchML does not sense the network state. PANAMA and ATP only react to congestion according to end-to-end signals without awareness of in-switch congestion state, resulting in delayed congestion feedback and control. (2) These protocols couple the congestion control of in-network aggregator resources and link bandwidth resources together, making it hard to allocate the two resources independently. Taking ATP as an example, each worker sends all gradients in its congestion window to the switch aggregator. The gradients expected to be aggregated at the switch randomly preempt the aggregators, easily resulting in an unreasonable allocation of aggregator resources and low efficiency of aggregation. (3) The straggler problem in shared clusters inevitably occurs due to resource competition. However, the existing protocols cannot identify the straggling worker and mitigate the straggler problem, leading to high aggregator occupancy and low aggregation throughput.

To address these challenges and provide efficient aggregation, we propose Aggregator-aware in-network Aggregation Transmission Protocol (A^2TP) based on the following insights. If the congestion control of in-network aggregator resources and network bandwidth resources are decoupled, the aggregator and bandwidth can be allocated independently to precisely control congestion at switch aggregator and ensure the fair or weighted fair allocation of in-network aggregator resources among multiple jobs. Moreover, if the aggregator resources are allocated to jobs according to their different straggling degrees, the impact of straggler could be alleviated, improving the aggregation efficiency.

Specifically, A^2TP infers congestion state of in-switch aggregator and bottleneck link according to hash collision at aggregator and Explicit Congestion Notification (ECN), respectively. Meanwhile, two window-based congestion control mechanisms are designed to respond to the two types of congestion signals, so as to realize the decoupled congestion control of aggregator and bandwidth resources. Moreover, to address the low efficiency issue under straggler, A^2TP first quantifies the straggling degree of job according to the reception rate of aggregated gradients at the worker. When the in-switch congestion occurs, the workers will tune the in-network congestion windows to allocate more switch aggregators to the leading job, thus reducing the aggregator occupancy by the straggling job and improving the total aggregation throughput.

In summary, we make the following contributions:

- We conduct experimental study to investigate the poor performance of the state-of-the-art INA protocols under heterogeneous scenarios. We reveal that the coupled congestion control of in-switch aggregator and bottleneck link, together with the straggler-oblivious manner in aggregator allocation lead to low aggregation efficiency.
- We propose a new end-to-end INA transmission protocol A^2TP, which uses two congestion windows to decouple the allocation of aggregators and bandwidth, and assigns aggregators in accordance with the straggling degree for multiple jobs, thus effectively alleviating the impact of straggler job on aggregation process.
- We implement A^2TP at P4-programmable switch and a kernel bypass protocol stack at the end host. We evaluate A^2TP in a hardware testbed across a large number of scenarios with realistic traffic patterns, workloads and training jobs. The results show that A^2TP reduces training time by up to 66% compared with the state-of-the-art INA protocols.

2 Background

2.1 Distributed Machine Learning

With the rapidly growing complexity and scale of ML models, the training time is also greatly increasing. To speed up training, DML is widely adopted by practitioners. The most commonly used DML technique is data parallel training, which divides dataset into multiple subsets and assigns a different subset to each worker. Specifically, each worker maintains a same model and needs to aggregate gradients in each training iteration to synchronize model parameters for all workers, ensuring convergence accuracy. Recently, parameter server (PS) [11] and Ring-AllReduce [2] communication patterns are generally used to aggregate gradients. In PS pattern, all workers synchronously send gradients to one or more specified PS servers for aggregation, and then the aggregated gradients are broadcasted back to all workers. In Ring-AllReduce pattern, all workers are deployed over a logical ring topology. Each worker accumulates the gradient sent by the previous worker with its own gradient and sends it to the next neighbor worker until one worker receives the aggregated gradient of all workers. Then the aggregated gradient is broadcast to other workers.

However, both PS and Ring-AllReduce generate a large amount of aggregation traffic across multiple workers, causing network congestion and making communication become the training bottleneck. For example, in DeepLight [12] model, the communication time accounts for 97% of the total time in each iteration at 10Gbps [4].

2.2 In-network Aggregation

Programmable switches (e.g., Intel Barefoot Tofino [7]) support the application services to offload some computing functions to the switch. The programmable switch exposes memory as stateless metadata and stateful registers to store the

needed states for calculations. Based on the programmable switch, a series of INA protocols are proposed, which offload gradient aggregation to the switch and use registers to aggregate gradients from different workers. These INA protocols effectively reduce the volume of aggregation traffic and alleviate network congestion.

Specifically, each worker divides gradients into fixed-sized pieces, which are assigned different sequence numbers. At the switch, the registers are organized as an array of aggregators, each of which contains an index. The gradients with the same sequence number from different workers are assigned the same index and thus aggregated in the same aggregator. Besides, the packets will be dropped after the gradients are accumulated in the aggregator. After all gradients with the same sequence number have been aggregated in the aggregator, the aggregation completes and the aggregated result is broadcasted back to all workers (in SwitchML [4], PANAMA [5]) or sent to PS (in ATP [6]). To provide the reliability of aggregation transmission, existing INA protocols regard the aggregated packet as an acknowledge (ACK), and the gradient with the next sequence number will be driven after the worker receives the ACKs in order.

Among the INA mechanisms, SwitchML and PANAMA offload all aggregation function to the switch and statically partition aggregator resources for multiple training jobs. Meanwhile, the BDP of each worker is limited to less than the amount of allocated aggregators, guaranteeing the correctness of aggregation. The recent work ATP sends gradients to the aggregator with best-effort to improve aggregator utilization. The gradients from multiple jobs are aggregated in the aggregator of the corresponding index according to random hash values. If the aggregator has been occupied by other gradients with different sequence numbers, ATP enables the current gradient to be aggregated at the PS.

3 Observation and Insight

The end-to-end aggregation protocol ensures the correctness of aggregation and supports the sharing of network resources among multiple training jobs. However, under heterogeneous scenarios, the existing aggregation protocols essentially suffer from the following problems. (1) Incorrect or delayed reaction of congestion control. (2) Failure of fair or weighted fair allocation of in-network aggregator resources among multiple jobs. (3) The high aggregator occupancy by straggler jobs and low end-to-end aggregation efficiency.

3.1 Experimental Observations

Aiming to provide reliable end-to-end aggregation services, existing aggregation protocols regard the network as a black box, while ignoring the in-switch congestion state and thus possibly shielding the fundamental causes of the above problems. In this section, fine-grained and multi-variable experimental observations are conducted to expose the root causes of these phenomena and problems under heterogeneous scenarios.

Straggler problem is common in shared clusters with heterogeneous resources [13, 14]. Firstly, though workers in the same job tend to be deployed in the same multi-GPUs machine, however, the large number of jobs have various requirements of GPU usage, resulting in the fragmentation of GPU resources [15]. It is inevitable that the workers of multiple jobs sit in the same machine, and they will compete for one NIC bandwidth [16, 17]. In shared clusters such as EC2 and Azure, Ref. [18] observes that the varies workloads lead to more than 2x throughput variance among different workers, resulting in volatile aggregation performance. Ref. [16] reports that the training performance shows up to 47% slowdown when running multiple 2-GPU ResNet-50 jobs on multiple 4-GPU servers. Furthermore, as a single server supports more GPUs (e.g. NVIDIA DGX [19] with 8 GPUs), it may be shared by more tenants. Such server may suffer from severe bandwidth degradation. For example, Ref. [20] shows that, when multiple tenants run communication intensive jobs, each job only reaches ~30Mbps with 1Gpbs NIC. The same problem is applied with the higher bandwidth 10-100Gbps NIC [21]. Besides, as practitioners usually upgrade computing devices in incremental manner, the shared clusters contain different types of GPU [8], leading to heterogeneous computation resources and straggling workers.

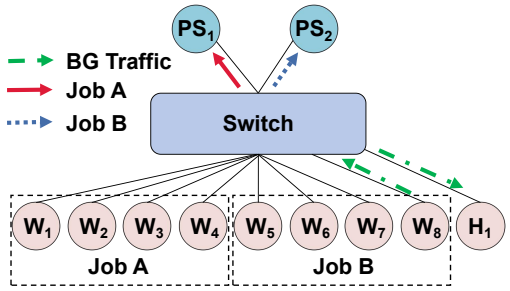
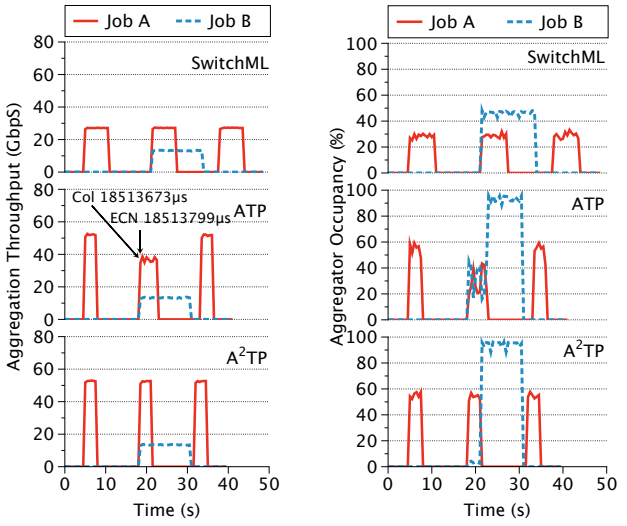


Figure 1. Topology.

To investigate the performance impact of straggler issue on existing aggregation protocols, we conduct a testbed experiment under a heterogeneous scenario. Figure 1 shows the test topology, which is a single rack structure. We adopt 11 Dell T5820 servers connected by a Tofino programmable switch. All links have 100Gbps bandwidth. There are 8 workers $\{W_1 \sim W_8\}$ and 2 PSs $\{PS_1, PS_2\}$. We run two aggregation jobs A and B, which are allocated at $\{W_1 \sim W_4, PS_1\}$ and $\{W_5 \sim W_8, PS_2\}$, respectively. Each worker repeatedly transfers 4MB gradients and has no computation phase, similar to micro-benchmark job in [6]. Job A has three iterations, each of which aggregates 21GB gradients. Job B has only one iteration with 21GB gradients and overlaps the 2nd iteration of job A. We set the number of switch aggregators to

450, according to the definition of peak throughput aggregators (PTA) of a single job in ATP. That is, the aggregation throughput with single job decreases when the number of aggregators is less than 450, and hardly increases when the number of aggregators is larger than 450. Thus the aggregator becomes a bottleneck when job A and B coexist.



(a) Aggregation Throughput (b) Aggregator Occupancy

Figure 2. Performance of different INA protocols under bandwidth contention.

We test the performances of SwitchML and ATP, and record the real-time aggregation throughput, which is the total bytes of gradients received per second at each worker. We simulate the heterogeneous bandwidth scenarios by generating background traffic from W_8 to server H_1. Thus, the sending rate of gradients from W_8 is suppressed to about 13Gbps. Figure 2(a) shows the results under heterogeneous bandwidth. The aggregation throughput of straggling job B is only about 13Gbps, which is limited by the sending rate of straggler W_8. When using SwitchML, the aggregators are allocated equally between job A and B. Thus, job A only reaches half of the maximum aggregation throughput in all iterations, regardless of whether job B transmits the gradients. Contrary to the static memory allocation in SwitchML, ATP makes best-effort use of aggregators, and thus enables job A to reach the maximum aggregation throughput of 53Gbps when job B does not transmit aggregation traffic. However, when jobs A and B coexist, due to the bandwidth contention, the aggregation throughput of job A decreases by 28%, as shown in Figure 2(a). In a nutshell, both SwtichML and ATP suffer from throughput loss due to the straggler job. Note that, small packet size (e.g. ~300B in SwitchML, ATP and A^{2}TP) brings additional packet processing overhead, resulting in a throughput limitation of 53Gbps for a single job.

3.2 Problem Insights

To understand the throughput loss problem in above experiment, we also record the real-time occupancy ratio of aggregators at the switch, as shown in Figure 2(b).

1) Unawareness of aggregator congestion leads to incorrect and delayed congestion control behavior. Existing aggregation protocols employ the end-to-end congestion control according to the ECN signal and packet loss. Specifically, SwitchML statically partitions aggregator resources according to the number of jobs and allocates more aggregator resources than the worker's BDP to ensure the correctness of aggregation. In a multi-job scenario, however, even if some jobs are in the computation phase without transmitting aggregation traffic, the remaining jobs cannot grab the idle aggregators to further reduce the data volume of model updates. As shown in Figure 2(b), in the first and third iterations of job A, the aggregation occupancy is only 30%.

The dynamic INA solution ATP fully utilizes switch aggregators with best-effort, and lets gradients to be aggregated at the PS when the aggregators are unavailable. However, when most in-switch aggregators are exhausted, a portion of gradients cannot be aggregated, possibly resulting in congestion at bottleneck link. Since ATP only senses network congestion via end-to-end signals without awareness of in-switch congestion state, the worker does not react until it receives the ECN signals from the congested bottleneck link. For example, we record the time when the sender receives the first aggregator collision and ECN feedback. As shown in Figure 2(a), the sender receives the first aggregator collision signal 126μs earlier than the first ECN feedback.

In summary, the static INA solutions cannot adjust the sending rate according to the usage state of the aggregator resources, easily leading to aggregator under-utilization for dynamic traffic. Though making better use of aggregators, the best-effort solution cannot timely react to aggregator congestion as only using the end-to-end congestion signals.

2) The coupled congestion control of in-switch aggregator and link bandwidth cannot provide fair or weighted fair resource allocation among multiple jobs.

Generally, multiple jobs have different requirement for resource allocation of in-network aggregator. To provide fairness among training jobs, the aggregators should be fairly shared by all jobs. However, to improve the aggregation efficiency, the straggler job should be allocated less aggregators than non-straggler ones. The reason is detailed in Section 3.2.(3). Nevertheless, since the existing INA protocols couple the congestion control of in-switch aggregator and link bandwidth resources, the two types of resources cannot be allocated independently among multiple jobs. For instance, ATP sends all gradients in the congestion window to the switch aggregator. The gradients of all jobs randomly preempt the aggregators and cannot provide fair or weighted fair allocation for multiple jobs. As shown in Figure 2(b),

under ATP, the instantaneous aggregator occupancy of job *A* and job *B* violently oscillates from a microscopic view, making it unable to converge to fair sharing, let alone the weighted fairness according to the job demands.

3) The straggler-oblivious manner causes the high aggregator occupancy.

When an aggregator is occupied by a gradient, the other gradients cannot use the aggregator until the current gradient completes aggregation. Thus, the aggregators occupied by the gradients from leading workers have to wait for the stalled gradients from stragglers, resulting in the long occupation time and low utilization of aggregators. If one worker becomes straggling due to background traffic, the leading workers in this job will still send all gradients in their congestion window to grab the aggregators, resulting in high space occupancy of aggregators. Even if the congestion window of the leading workers is small, the gradients in the aggregators will wait for the straggling gradients, leading to long occupancy time at aggregators and low aggregation efficiency. Therefore, to improve aggregation efficiency, the straggler jobs should relinquish their high occupancy of aggregator to the non-straggler ones under the contention at in-switch aggregators.

However, existing INA protocols cannot sense the straggling degree, and treat all training jobs in the same way. Taking SwitchML as an example, the aggregator resources are fairly allocated for multiple jobs no matter the jobs are stragglers or not. As Figure 2(b) shows, though most aggregators used by the straggler job *B* keep waiting for long time, it still occupies almost half of total aggregators. Thus, without enough aggregators, it is hard for job *A* to reduce the more volume of traffic and mitigate congestion of bottleneck link.

4 A^{2}TP Design

4.1 Basic Idea

To address above problems, we propose following two basic ideas to improve the efficiency of existing INA protocols.

1) Decoupling the congestion control of in-network aggregators and link bandwidth resources. As the ECN signal cannot accurately reflect the congestion state at in-switch aggregator, we propose a new congestion signal to reflect the aggregator congestion, thus detecting the congestion in time. Meanwhile, to provide the fair or weighted fair allocation of in-network aggregators and allocate resources more efficiently for multiple jobs, we design the decoupled mechanisms to control congestion of in-network aggregator and link bandwidth.

2) Assigning aggregator resources in accordance with the straggling degree of each job. Firstly, since the job straggler is common in heterogeneous scenarios, and the straggling degree of individual job varies dynamically, we need to measure and quantify each job's straggling degree in real time. Besides, the congestion control mechanism is required to proactively adjust the number of used aggregators according to the straggling degree of each job.

4.2 A^{2}TP Overview

A^{2}TP is a window-based INA transmission protocol. The workers send gradients to the switch and make best effort to use in-switch aggregators via hash function. If hash collision occurs, the gradients will be forwarded to the PS. Otherwise, the gradients occupy the aggregator and wait for gradients from the other workers. After the in-switch aggregators finish aggregation, the aggregated results are sent to the PS, which broadcasts aggregated gradient back to all workers. A^{2}TP regards the aggregated gradient as the ACK and uses congestion signal carried by ACK to tune the congestion window at the worker. Compared with current INA transmission protocols, A^{2}TP has two following features: (a) A^{2}TP re-designs the congestion control mechanism, decoupling the congestion control of in-network aggregators and link bandwidth to independently allocate two types of network resources for each job. (b) A^{2}TP estimates the straggling degree of each job, and employs the straggler-aware congestion control to mitigate the impact of straggler jobs. Specifically, A^{2}TP consists of two components:

1) Hybrid signals. A^{2}TP adopts multi-dimensional signals to guide the adjustment of each worker's congestion window. First, the ratio of hash collision at aggregator is chosen as a congestion signal to indicate the in-switch congestion state. Second, similar to ATP and PANAMA, A^{2}TP uses ECN as the congestion signal of bottleneck link. Besides, the workers estimate the real-time straggling degree according to the numbers of sent gradients and received aggregation ACKs in each RTT.

2) Decoupled congestion control. A^{2}TP maintains two congestion control loops, including the aggregator congestion control (ACC) and the link congestion control. They separately adjust two congestion windows according to congestion state in different types of resources. Specifically, ACC regulates an aggregator congestion window (ACW), which is the number of gradients sent to the aggregator. ACW is adjusted according to the level of aggregator congestion and the straggling degree of the current job. Besides, for link congestion control, A^{2}TP adjusts the link congestion window (LCW) according to the fraction of ECN marks, similar to DCTCP [22].

We use an example to illustrate how A^{2}TP decouples the congestion control of in-switch aggregators and link bandwidth resources, and assigns different number of aggregators in accordance with their straggling degrees. As shown in Figure 3, two jobs *A* and *B* start at the same time, and each job consists of two workers and one PS. We assume that *B* is a straggling job. Each worker maintains an aggregator congestion window *ACW* and a link congestion window *LCW*. The initial sizes of both congestion windows are 3, and the

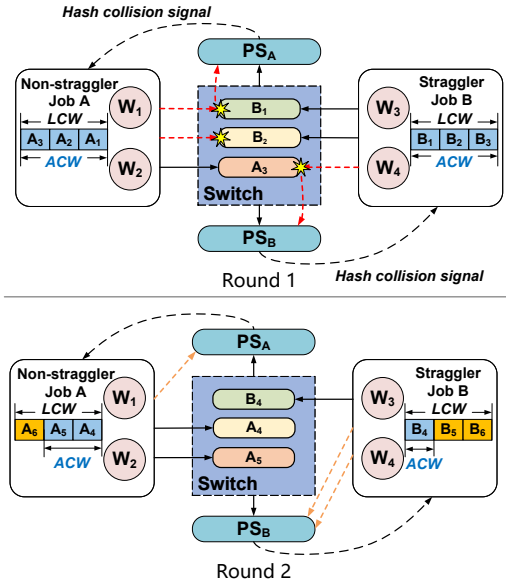


Figure 3. Examples of congestion control behavior in A^{2}TP.

switch has only 3 aggregators. In the first round, all 6 gradients from two jobs are sent to aggregators, resulting in hash collisions. After detecting the hash collision, the workers cut the *ACW*s. Since job *B* is the straggler, it aggressively reduces its *ACW* to 1, while job *A* gently shrinks its *ACW* to 2. Therefore, in the next round, the total number of gradients aggregated at the switch is reduced to 3, avoiding random hash collisions. Meanwhile, the ratio of aggregator occupancy by the straggling job *B* is reduced, improving the aggregation efficiency. Finally, no matter the gradients are aggregated at the switch or not, all gradients will be injected into the bottleneck link. Thus, we use *LCW* to control the total number of in-flight traffic to avoid congestion at the bottleneck link.

4.3 Design Detail

4.3.1 Hybrid Signals. A^{2}TP tunes the sending rate according to multi-dimensional signals including hash collision, ECN feedback and straggling degree.

Hash collision: In current INA transmission protocols, the traditional end-to-end signals such as delay, packet loss and ECN cannot timely and accurately reflect the congestion state of the in-switch aggregators. To collect the precise congestion state at the aggregators with low overhead, we propose to use the aggregator hash collision as the congestion signal. The hash collision occurs when the different gradients are hashed into the same aggregator. If the aggregators are sufficient, the possibility of hash collision is low. Otherwise, the aggregator congestion will lead to frequent hash collisions. Therefore, A^{2}TP uses the frequency of hash collision to indicate the congestion level of in-network aggregator.

In ATP, when a gradient arrives at the aggregator and experiences hash collision, it will be marked as collision. However, the gradient having occupied the aggregator will not be marked and thus the hash collision cannot be detected, resulting in under-estimation of congestion state. To address this issue, A^{2}TP makes a slight change to ATP header format and switch memory layout by adding 1bit field (Col) to record hash collisions. Thus, each aggregator in A^{2}TP consists of 32bits bitmap that tracks the aggregation workers, 32bits counter of aggregation times, 2bits ECN and Col fields, 32bits job ID and sequence number, 32bits timestamp, and 248Bytes aggregated value. The Col fields in both packet header and switch memory are initially set to 0. When a gradient occupies an aggregator and other gradients are hashed to the same aggregator, the aggregator's Col field is updated to 1. After the aggregation completes, the Col field of the aggregated gradient is set to 1 and is finally broadcasted back to all workers by using the ACK packets. Thus, as long as a hash collision occurs on the aggregator, the job occupying the aggregator also receives the hash collision signal.

Finally, the aggregator congestion level α_i of ith RTT is quantified as:

$$\alpha_i = (1 - w) \times \alpha_{i-1} + w \times h_i, \tag{1}$$

where h_i is the fraction of gradients that were marked with *Col* bit in recent *ACW*, and $0 < w < 1$ is the weight factor given to new samples. If the aggregator congestion is light, the ratio of hash collision is low and α_i is close to 0. Otherwise, α_i is close to 1 when aggregator congestion is in high level.

ECN feedback: A^{2}TP adopts ECN as the link congestion signal. At each switch no matter it is the aggregator switch or not, if the queue occupancy is greater than the ECN marking threshold, the arriving gradient is marked with the CE codepoint. After the aggregation completes, the ACK packets with CE marked are broadcasted back to all workers, so that all workers synchronize the adjustment of window to mitigate network congestion.

Specifically, A^{2}TP updates link congestion level factor β_i of ith RTT as:

$$\beta_i = (1 - g) \times \beta_i + g \times e_i, \tag{2}$$

where e_i is the fraction of packets marked ECN in the recent *LCW* and g is a weight factor.

Straggling degree: When a worker becomes straggling, the gradients from leading workers will occupy the aggregators and wait for the stalled gradients at the switch or PS. Only the gradients having completed aggregation will trigger the ACK packets. Thus, the workers estimate the straggling degree of current job according to the numbers of sent gradients and received ACKs.

Specifically, at the end of ith RTT round, A^{2}TP records the size of Link Congestion Window (LCW) LCW_i and the number of ACKs n_i received in the ith RTT. A^{2}TP quantifies

the straggling degree as:

$$\gamma_i = \frac{n_i}{LCW_i}, \tag{3}$$

where $0 <= \gamma_i <= 1$. When there is no straggler, γ_i is equal to 1, indicating that all gradients transmitted in the previous round have been aggregated. If γ_i approaches 0, it indicates that the job experiences seriously straggling, and most of gradients in last LCW persistently occupied the aggregators.

4.3.2 Decoupled Congestion Control.

Compared with the current INT designs adjusting congestion window based solely upon the end-to-end signal, A²TP decouples the congestion control of in-network aggregator and link bandwidth according to multiple signals. At the workers, the Aggregator Congestion Window (ACW) is adjusted according to the fraction of hash collision and straggling degree, while the Link Congestion Window (LCW) is adjusted in accordance with the fraction of packets that are ECN marked.

Aggregator congestion control: Each worker maintains its ACW to regulate the number of gradients aggregated at the switch. Since the packets aggregated at the switch are still a portion of the in-flight packets, ACW is a part of LCW. The gradients inside ACW will be aggregated at the switch, while other gradients inside LCW will be directly sent to PS for aggregation.

The size of ACW is updated once for every LCW data (roughly one RTT). To achieve fair allocation of aggregator among multiple jobs (with same priority), A²TP adopts Additive Increase and Multiplicative Decrease (AIMD) strategy to adjust ACW in accordance with aggregator congestion level. Due to the random hash, the collisions cannot be avoided even if the number of in-switch aggregators is greater than the number of gradients aggregated at the switch. If the congestion control is too sensitive to hash collision, the aggregator maybe under-utilized due to unnecessary reduction of sending rate. Thus, A²TP uses an aggregator congestion threshold H. Only if α_i exceeds H, it is indicated that the in-switch aggregator becomes congested.

We define the aggregator congestion degree p_i as $p_i = \frac{\alpha_i - H}{1-H}$. If α_i is less than H, p_i is set to 0. When α_i is only slightly greater than H, the aggregator congestion is in low level, and p_i approaches 0. On the contrary, p_i is close to 1.

When $p_i > 0$, the worker cuts its ACW according to the decreasing factor d_i as follows:

$$ACW_{i+1} = ACW_i \times (1 - \frac{d_i}{2}), \tag{4}$$

where d_i is determined by the aggregator congestion level p_i and straggling degree γ_i as:

$$d_i = p_i^{\gamma_i}. \tag{5}$$

The relation between d_i, p_i and γ_i is shown in Figure 4. When the aggregator congestion is in very high level, d_i is near 1. ACWs of both the straggler and non-straggler jobs will be halved by maximum. For the straggling degree, γ_i close to 0 indicates large, and γ_i close to 1 indicates small straggling degree. In the former case, d_i is close to 1, and ACW will be halved at most. In the latter case, d_i approaches to p_i, and ACW decreases in accordance with p_i. Thus, the straggler jobs will release the aggregator resources for the non-straggler jobs, improving the aggregation throughput.

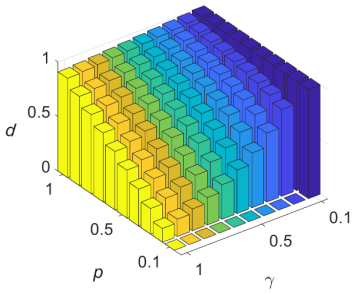


Figure 4. The decreasing factor d of ith RTT.

Link congestion control: To resolve the congestion at bottleneck link, A²TP adopts the link congestion control strategy similar to DCTCP [22]. When congestion occurs, LCW is reduced as follows:

$$LCW_i = LCW_i \times (1 - \frac{\beta_i}{2}). \tag{6}$$

Note that if the level of link congestion is higher than that of aggregator congestion, LCW may be smaller than ACW, meaning that the link bandwidth becomes bottleneck of INA. In this case, the size of ACW should be caped by LCW. Therefore, ACW is set as $min(ACW, LCW)$.

Since we design a DCTCP-like mechanism to adjust the rate of training traffic, when competing for bandwidth with regular traffic (which commonly adopts DCTCP in datacenters), the bandwidth allocation among different connections will converge to fairness.

4.4 Model Analysis

We next analyze the steady-state behavior of A²TP's aggregator control loop in several simplified settings. First, since hash collisions rarely occur after the warm-up phase in dynamic hashing scheme [6], we assume that hash collisions do not occur until all M aggregators are occupied. Then, we consider N long-lived flow from multiple non-straggling jobs with same round-trip time RTT. We assume that ACWs of N flows are synchronized and follow same sawtooths. Thus the number of hash collisions at time t is given by:

$$C(t) = N \times ACW(t) - M, \tag{7}$$

where $ACW(t)$ is the aggregator window size of a single flow. $C(t)$ follows the identical sawtooths, which are quantified as the maximum number of hash collisions (C_{max}), the amplitude of collision oscillations (A), and the period of oscillations [22].

Thus, when the straggling degree is 1, the decreasing factor d is calculated by dividing the number of gradients (or packets) sent to aggregator in the last RTT by the total number of gradients sent in a full period of the sawtooth.

For a single flow, we define $G(ACW_1, ACW_2)$ as the number of gradients sent by this flow while its ACW increases from ACW_1 to ACW_2. Therefore, it takes $ACW_2 - ACW_1$ RTTs and the average window size is $(ACW_1 + ACW_2)/2$. Thus, we get $G(ACW_1, ACW_2)$ as follows:

$$G(ACW_1, ACW_2) = \frac{ACW_2^2 - ACW_1^2}{2}. \tag{8}$$

Let $ACW^* = \frac{M}{(1-H)N}$, which is the boundary value that starts the aggregator congestion control. Due to the response time of one RTT, the aggregator window size continues to increase by 1 gradient to $ACW^* + 1$. Therefore, we get:

$$d = \frac{G(ACW^*, ACW^* + 1)}{G((ACW^* + 1)(1 - \frac{d}{2}), ACW^* + 1)}. \tag{9}$$

With Eq.(8) and Eq.(9), we get:

$$d^2(1 - \frac{d}{4}) = \frac{2ACW^* + 1}{(ACW^* + 1)^2} \approx \frac{2}{ACW^*}, \tag{10}$$

where the approximation is valid when $ACW^* >> 1$. We assume that d is small and thus $d \approx \sqrt{2/ACW^*}$. The amplitude of oscillation in ACW of a single flow S is given by:

$$S = (ACW^* + 1) - (ACW^* + 1)(1 - \frac{d}{2}). \tag{11}$$

When there are N flows, we get A as:

$$\begin{aligned} A = NS &= N(ACW^* + 1)\frac{d}{2} \approx \frac{N}{2}\sqrt{2ACW^*} \\ &= \frac{1}{2}\sqrt{\frac{2MN}{1-H}}, \end{aligned} \tag{12}$$

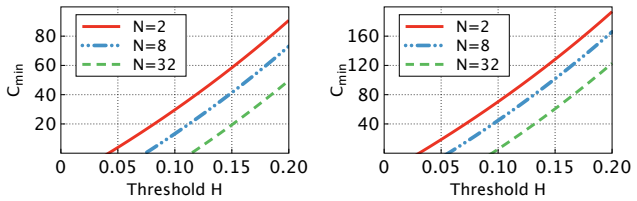
Next, according to Eq.(7), we get the maximum number of hash collisions as:

$$C_{max} = N(ACW^* + 1) - M = \frac{MH}{1-H} + N. \tag{13}$$

Plugging in Eq.(12) and Eq.(13), we get the minimum number of hash collisions as:

$$\begin{aligned} C_{min} &= C_{max} - A \\ &= \frac{MH}{1-H} + N - \frac{1}{2}\sqrt{\frac{2MN}{1-H}}. \end{aligned} \tag{14}$$

To fully utilize the in-switch aggregators, we need to choose H to make $C_{min} >= 0$. We calculate the lower bound of H according to Eq.(14) under varying number of aggregators and flows, so as to guide the selection of threshold H. As shown in Figure 5(a), as the number of flows increases, aggregator congestion control requires a larger H to ensure high utilization of aggregators. Specifically, when two flows are sharing 450 aggregators, H only needs to be greater than 0.03 to achieve high utilization, while for 32 flows competing at the switch aggregators, H has a lower boundary of approximately 0.12. Figure 5(b) shows the result under $M = 900$, the lower bound of H decreases. The lower bound of H is 0.09 under 32 flows. Since a too large H will lead to a high frequency of hash collision and make it hard to provide weighted fair allocation of in-network aggregator resources for multiple jobs, H is finally chosen to make $C_{min} = 0$. We analyze the sensitivity of H in Section 6.8.



(a) M=450 (b) M=900

Figure 5. Selection of threshold H.

5 Implementation

We implement the prototype of A^{2}TP using a kernel bypass protocol stack and P4-programmable switch on a real testbed.

End-host. We leverage Mellanox's RAW ETHERNET user-space network programming model to remove kernel-space overhead, similar to ATP. We use *ibv_exp_post_send* and *ibv_poll_cq* function to send and receive packets, respectively. The decoupled window adjustment mechanism works after receiving the ACK packets. Meanwhile, we distinguish the packets sent to switch and to PS server by assigning the value of *is_resend* in the packet header. Besides, for distributed training framework, we follow ATP's work, which integrates ATP with BytePS. BytePS sets up the context for workers to communicate with the PS. All we need to do is modifying the *Push*() operation to aggregate gradients by using A^{2}TP. Then, we use BytePS's plugin for distributed training with DL framework like Pytorch. To aggregate gradients from different workers, we register the *Push*() operation on every layer's parameters through the *register*() function provided by Pytorch. The *Push*() operation will be called after the corresponding gradients calculation of each layer's parameters is completed.

Switch. We implement the INA logic of A^{2}TP based on the Wedge 100BF-32X programmable switch with P4 language. The ingress pipeline for aggregation is shown in Figure 6, which contains multiple match-action tables. Specifically, the packet firstly matches the *Filter* table. If the *is_resend* flag in the packet header has been set as 1 at the end-host, it will be directly forwarded to the parameter server. Otherwise, it will trigger the aggregation and try to write the gradients to the aggregators. There are two kinds of packets forwarded

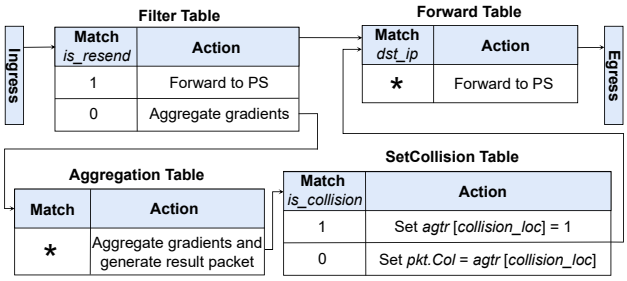


Figure 6. P4 implementation.

by *Aggregation* table. The first one is the packet having experienced hash collision and the other one is the aggregation result. In the *SetCollision* table, the register *agtr* at the location *collision_loc* will be set to 1 when the *is_collision* flag in packet header is 1, which indicates the packet has caused a hash collision in the aggregator. If the packet is an aggregation result (with *is_collision* flag as 0), its *pkt.Col* flag will be set according to the value of register *agtr* at the location *collision_loc*. The *pkt.Col* flag will be used as the signal of aggregator hash collision to tune the sending rate of workers. Finally, both the aggregation result and packets having experienced collision will be forward to parameter server according to the route in *Forward* table.

6 Testbed Evaluation

6.1 Experimental Setup

Testbed. Our testbed consists of a Barefoot Wedge 100BF-32X programmable switch and 11 GPU servers. All servers are connected to the switch with 100Gbps links. Each server is equipped with a Mellanox ConnectX5 100Gbps NIC, an Intel Xeon W-2255 CPU (3.7GHz, 20 logical processors and 19.75MB Cache), 64GB DDR4 DRAM and an NVIDIA RTX3090 GPU.

Workloads. We choose VGG16 [23], ResNet50 [24], AlexNet [25] on the image dataset CIFAR10 [26] and Bert [27] on the corpus dataset multi30k [28] as the representative models. VGG16 (528MB model size) and ResNet50 (98MB model size) are selected as the communication-intensive and computation-intensive workloads in most experiments, respectively. We adopt SGD as the optimizer and set the initial learning rate to 0.01.

Straggler injection. We generate background traffic with varying sending rates on the workers to test the impact of stragglers with varying straggling degree. To simulate the scenarios where multiple jobs are assigned to the same server and share the NIC bandwidth, the sending rates of background flows are set according to the training traces of DNN model.

Baselines. We compare A²TP with SwitchML and ATP with two PSs. Note that, we only use N workers and 1 PS mode for ATP and A²TP in the evaluation. For colocated-PS mode [29], the worker and PS on the same server will communicate through loopback traffic, alleviating the communication overhead. However, ATP and A²TP will still benefit from colocated-PS, since the traffic volume from other workers to the colocated-PS is reduced by in-network aggregation. Besides, according to empirical studies, the parameters both w and g are set to 1/16 in A²TP.

Metrics. We evaluate A²TP on four performance metrics: (1) Aggregation throughput is the total size of gradients aggregated in unit time. (2) Aggregator occupancy is the fraction of used aggregators to all aggregators. (3) Time to Accuracy (TTA) is the real-time curve of training accuracy changing with time. (4) Iteration Time is the average iteration time of gradient aggregation for multiple rounds of training iterations.

6.2 Aggregation Throughput and Aggregator Occupancy

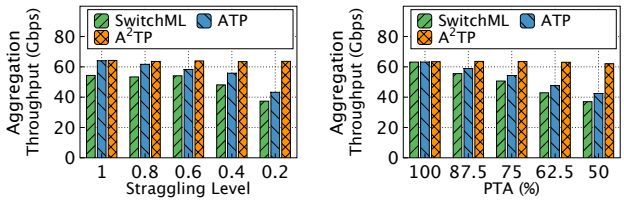
Based on the experimental setup in Section 3.1, we measure aggregation throughput and aggregator occupancy to compare the performances between A²TP, SwitchML, and ATP. As shown in Figure 2, when using A²TP, since the aggregator resources are reasonably allocated for multiple jobs, the aggregation throughput of job A is as high as 52Gbps, and the aggregator occupancy of job B is greatly suppressed when two jobs coexist. Specifically, after detecting the aggregator is congested, A²TP combines the straggling degree estimation with aggregator congestion control to reduce the aggregator congestion window of job B more aggressively, reducing the aggregator occupancy of straggler job B and improving the aggregation efficiency. Due to the decoupling of the congestion control of in-network aggregators and link bandwidth, the straggler job B abandons most of aggregator resources and utilizes available link bandwidth to directly upload gradients to PS. In contrast, the non-straggler job A makes almost full use of aggregator resources to maintain a high aggregation throughput.

6.3 Varying Straggling Level and Number of Aggregators

We evaluate the performance of A²TP with varying straggling level. We start two micro-benchmark jobs A and B synchronously over $\{W_1 \sim W_4, PS_1\}$ and $\{W_5 \sim W_8, PS_2\}$, respectively, and measure the total aggregation throughputs.

Firstly, we fix the number of aggregators at 450 and vary the straggling level of job B from 1 to 0.2, where the straggling level is defined as the ratio of available bandwidth of straggling worker over the leading worker. Specifically, for straggling level of 1, job B is a non-straggler job, while for straggling level of 0.2, the worker W_8 has only 20% of available bandwidth due to network contention.

Figure 7(a) shows that, the aggregation throughputs of the dynamic solutions (A²TP and ATP) are higher than that of the static solution (SwitchML) in all cases. This is because the



(a) Straggling Level (b) Number of Aggregators

Figure 7. Performance of A²TP under varying situation.

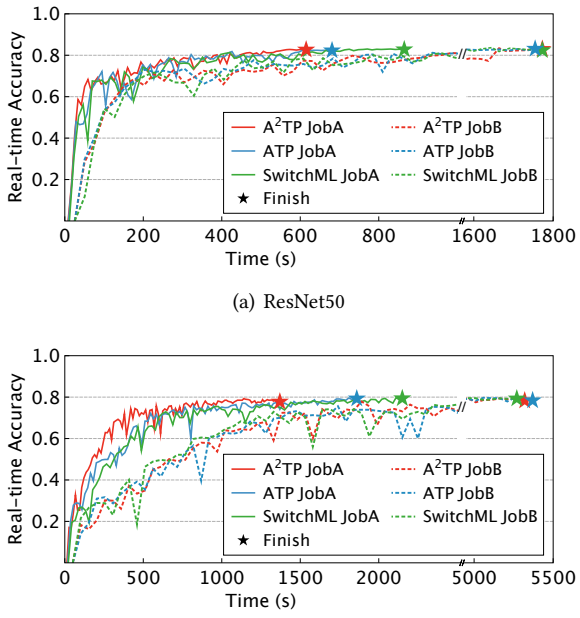
dynamic solutions effectively share the unused aggregators across jobs and support PS-based aggregation. Besides, the aggregation throughputs of SwitchML and ATP decrease with increasing straggling level, while A²TP still maintains the high throughput. The reason is that A²TP reasonably allocates aggregator resources for multiple jobs when the aggregator is congested. The non-straggler job *A* occupies more aggregators than job *B*, improving the aggregation efficiency.

Moreover, we fix the straggling level of job *B* and vary the number of aggregators from 100% to 50% of Peak Throughput Aggregators (PTA) for the two jobs, where PTA is the minimum number of aggregators required to achieve the maximum aggregation throughput. In this case, PTA is 900 aggregators. Figure 7(b) shows that, when using 100% of PTA, the aggregation throughputs of the three protocols are similar due to sufficient aggregators. When the number of available aggregators decreases, the aggregation throughput of A²TP is kept at 63Gbps, while those of SwitchML and ATP gradually decrease. Specifically, A²TP increases the aggregation throughput by up to 68% and 46% than that of SwitchML and ATP, respectively.

6.4 Training Efficiency

We evaluate the training efficiency of different solutions under the influence of stragglers. Note that we use real-time accuracy and time to maximum accuracy as the metrics of training efficiency. In this test, we use ResNet50 and VGG16 as the training models, where ResNet50 has less communication time due to fewer parameters than VGG16. There are 450 aggregators at the switch. The non-straggler job *A* running on the $\{W_1 \sim W_4, PS_1\}$ and the straggler job *B* running on the $\{W_5 \sim W_8, PS_2\}$ contend for the in-switch aggregators. Meanwhile, another VGG16 training model runs on worker W_8, making job *B* become straggling.

Figure 8 shows the real-time accuracy of SwitchML, ATP, and A²TP on two jobs. For SwitchML, ATP, and A²TP, the time to reach maximum accuracy for the straggler job is much larger than that for the non-straggler job. Fortunately, by decoupling congestion control of aggregator resources and links bandwidth resources, A²TP is sensitive to the straggler jobs, and actively assigns more aggregator resources to



(a) ResNet50

(b) VGG16

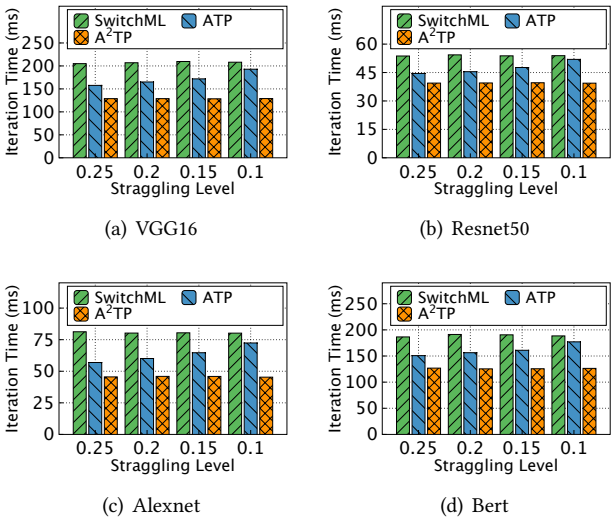
Figure 8. Time to Accuracy.

non-straggler jobs while directly sending gradients of straggler jobs to the PS. Thus, A²TP effectively reduces the hash collision at aggregator, improves the aggregation throughput and speeds up convergence of model training. Compared with SwitchML and ATP, A²TP reduces the convergence time by up to 36% and 26%, respectively.

6.5 Varying Workload

We evaluate A²TP performance under four training models including VGG16, Resnet50, Alexnet and Bert. Under each training model, there is one non-straggler job and another straggler job. The parameter settings are same as those of Section 6.4. We vary the straggling degree by changing the sending rate of the background flow according to the total throughput of the training job.

Figure 9 shows the average iteration time of non-straggler job for SwitchML, ATP, and A²TP by changing the straggle level from 0.25 to 0.1. The average iteration time of ATP increases with the heavier straggle degree, since ATP cannot perceive the straggler and the straggler job occupies the aggregator resource for a longer time. Compared with SwitchML and ATP, the iteration time of A²TP is consistently small. This is because A²TP is sensitive to straggler and releases the resources occupied by stragglers to non-straggler. Compared with SwitchML and ATP, A²TP reduces iteration time by up to 44% and 37%, respectively. Since A²TP alleviates the impact of straggler jobs on the other non-straggler ones, it obtains more benefits under serious straggler scenario across different training models.



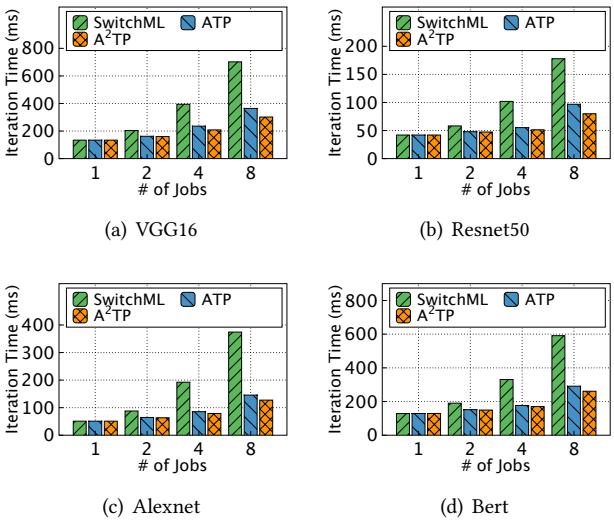
(a) VGG16 (b) Resnet50

(c) Alexnet (d) Bert

Figure 9. Avg. iteration time under varying straggling level.

6.6 Large Scales

We evaluate the performance of A²TP in multiple-job scenarios by relaunching identical DNN training jobs. Specifically, we vary the total number of jobs from 1 to 8. Every job has 4 workers, each of which is evenly allocated to the servers. There are at most 4 workers of different jobs in a server. The aggregators used by SwitchML are equally divided according to the number of jobs.



(a) VGG16 (b) Resnet50

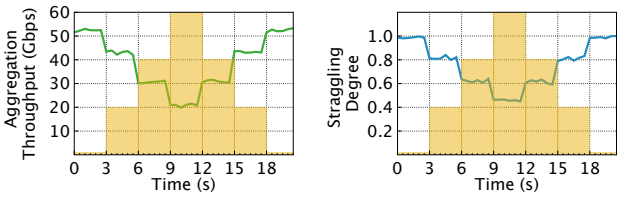
(c) Alexnet (d) Bert

Figure 10. Avg. iteration time under varying number of jobs.

Figure 10 shows the average iteration time of all jobs with different kinds and numbers of DNN training jobs. The iteration time of all schemes increase with more jobs due to the fewer available aggregators in each job. As shown in Figure 10, A²TP reduces the average iteration time by up to 66% and 18% compared with SwitchML and ATP with stragglers (i.e., the number of jobs is larger than 2), indicating that A²TP accelerates the aggregation progress of normal jobs. Besides, A²TP achieves high speedup with more jobs. When the number of jobs is 4, A²TP only reduces the average iteration time by 12% compared with ATP. However, A²TP achieves 18% speedup under 8 jobs because ATP has lower aggregation efficiency with more jobs.

6.7 Effectiveness of Straggling Estimation

The straggling estimation of A²TP aims to capture the straggling degree of the job timely and accurately. We evaluate the effectiveness of A²TP straggling estimation when a straggler exists. We set up a single micro-benchmark job across $\{W_5 \sim W_8, PS_2\}$. We add background traffic with varying strength of bandwidth competing on the link from worker W_8 to switch. The job and background traffic start at 0s. The strength of background traffic increases with 3s time interval until 12s, and the corresponding sending rate is 1Gbps, 20Gbps, 40Gbps and 60Gbps, respectively. After 12s, the background traffic gradually reduces the strength with 3s interval, and the transmission stops at 20s. We measure the real-time aggregation throughput and estimated straggling degree.



(a) Aggregation Throughput (b) Straggling Degree

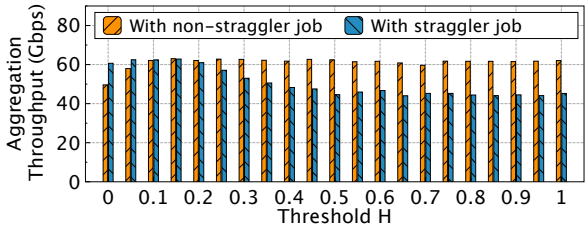
Figure 11. Effectiveness of straggling estimation.

Figure 11(a) shows the aggregation throughput of the job. As the strength of background traffic increases, the available bandwidth of W_8 decreases and the job straggling level increases, slowing down the aggregation throughput. On the contrary, as the strength of background traffic decreases, the aggregation throughput increases gradually. During this period, the estimated straggling degree has a similar trend with actual straggling level, as shown in 11(b). Specifically, when the aggregation throughput drops to 22Gbps due to straggler, the estimated straggling degree is 0.4, and if there is no straggler, the estimated straggling degree is approximately 0.99. This result indicates that, A²TP accurately estimates the straggling degree of job in real time, so as to precisely guide the adjustment of aggregator congestion window.

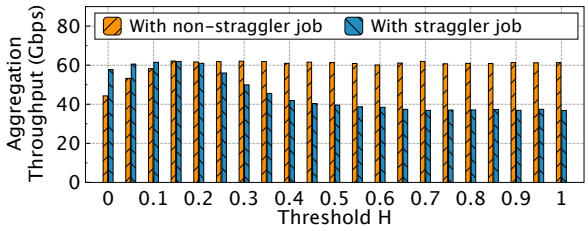
6.8 Sensitivity Analysis

The sensitivity analysis aims to evaluate the effect of threshold H on A²TP performance. The experimental setup and

topology are the same as in Section 6.3, where the number of aggregators for switches is fixed at 450. The experiments are conducted in two scenarios, namely with non-straggler job and with straggler job. Note that each job contains 4 workers and a parameter server. With straggler job, a half of jobs are non-straggler jobs, and others are straggler jobs, each of which has one straggling worker. We report the effect of threshold H on A^{2}TP's aggregation throughput with non-straggler job and with straggler job.



(a) N=2



(b) N=8

Figure 12. Aggregation throughput under varying number of jobs N and threshold H.

Figure 12 shows the total aggregation throughput under varying threshold from 0 to 1. With non-straggler job, the threshold H needs to be greater than 0.05 when N is 2, otherwise it will cause loss of aggregator utilization. This is because when the threshold H is too small (i.e., $H < 0.05$), the aggregator prematurely triggers aggregator congestion control, resulting in too aggressive congestion control. Moreover, with the increase of N, H also needs to increase to achieve full aggregator utilization, which follows the conclusion of the theoretical analysis. For example, when N is 8, H should be greater than 0.1 to achieve maximum aggregator utilization as shown in Figure 12(b).

Besides, A^{2}TP needs a reasonable threshold H to distinguish the straggling jobs and non-straggler jobs, and assigns them different aggregator resources. As shown in Figure 12(a), with a smaller threshold H as 0.15, A^{2}TP achieves about 60Gbps aggregation throughput with straggler. Besides, though a smaller threshold H works well in distinguishing two different jobs, it achieves low aggregation throughput due to low utilization (i.e., as shown in Figure 12(b), the aggregation throughput is only 57Gbps when H is 0, while the maximum throughput is 63Gbps when H is 0.15). With straggler jobs, when the threshold exceeds 0.2, the aggregation throughput decreases significantly, and A^{2}TP gets the lowest throughput when the threshold is larger than 0.55. This is because the extremely large threshold makes A^{2}TP have no chance to control the number of aggregators occupied by jobs with straggling worker, resulting in severe performance degradation. Therefore, we empirically set H to 0.15 to achieve both high aggregator utilization and weighted fair allocation of in-network aggregator resources.

6.9 Resource Overhead

We compare the resource overheads of SwitchML, ATP, and A^{2}TP including Match Crossbar, Hash Bits, Gateway, SRAM, VLIW Actions, and ALU Instruction, which are reported by the P4 compiler.

As shown in Table 1, since INA requires to frequently access a large number of registers for gradients aggregation, SwitchML, ATP, and A^{2}TP all consume many SRAM and ALU resources. ATP and A^{2}TP require less Match Crossbar and VLIW Actions than SwitchML as they handle packet loss and recovery at the end-hosts. Since A^{2}TP adds only 1bit of hash collision fields in the aggregator, the overhead of A^{2}TP is slightly higher than that of ATP. Therefore, A^{2}TP can be deployed in existing P4-programmable switch with tiny additional overhead.

Table 1. Overhead of different aggregation scheme.

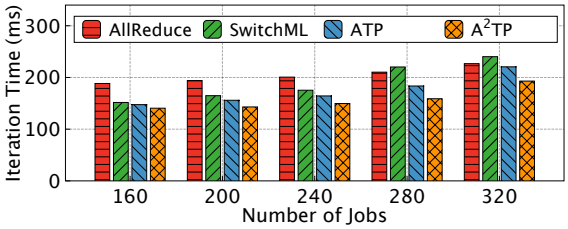
Resource	SwitchML	ATP	A^{2}TP
Match Crossbar	21.88%	14.13%	14.84%
Hash Bits	7.73%	6.77%	7.41%
Gateway	13.54%	50.00%	52.08%
SRAM	45.94%	44.79%	45.94%
VLIW Actions	28.39%	14.58%	14.84%
ALU Instruction	79.17%	75.00%	77.08%

7 Simulation Evaluation

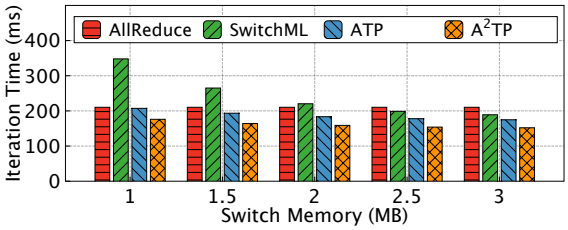
To evaluate the performance of A^{2}TP in the large-scale scenarios, we conduct NS3 simulations in the realistic datacenter topology and ML workloads. We use the leaf-spine topology with 12 ToR and 12 core switches. Each of ToR switches connects to 48 hosts and the whole network has 576 hosts. The bandwidth of all links are 100Gbps and the maximum base RTT is set to 16μs. We adopts ECMP for the load balancing.

We compare the performance of A^{2}TP with Ring-Allreduce, SwitchML and ATP. For all INA solutions, the packets size is set to 306B [6], which includes 58B header and 248B payload. Simlilar to SwitchML and ATP, A^{2}TP supports the extension of in-network aggregation to multiple switches. To support non-deterministic routing algorithm such as ECMP, all INA solutions only deploy aggregation function at ToR switches,

providing 2-level in-network aggregation for cross-rack traffic. The workloads are generated according to the training traces of ResNet50, AlexNet, VGG16 and LSTM [30] (1748MB model size). In our experiment, the proportions of the number of jobs for the four training models are same. By default, the switch memory for aggregation is 2MB, the number of generated jobs is 280, and the number of workers in each job is randomly and uniformly distributed in the range of 2-8. We randomly place the worker for each job, and a host may be shared by multiple jobs (no more than 8 jobs).



(a) Average iteration time under varying number of jobs



(b) Average iteration time under varying memory sizes

Figure 13. Average iteration time in different scenarios.

We evaluate the average iteration time of A^{2}TP under varying number of jobs from 160 to 320, and different switch memory from 1MB to 3MB. As shown in Figure 13(a), A^{2}TP outperforms the existing INA solutions and Ring-Allreduce under different loads due to the alleviation of low aggregation efficiency caused by straggler. Ring-Allreduce generates more network traffic than INA solutions, resulting in higher communication overhead. Specifically, compared with Ring-Allreduce, SwitchML and ATP, A^{2}TP reduces the average iteration time by up to ~26%, ~27% and ~13%, respectively. Figure 13(b) shows the performances of the four algorithms with varying switch memory. A^{2}TP achieves the lowest iteration time even when switch memory is only 1MB. As switch memory decreases, SwitchML suffers from significantly performance degradation, since the aggregator resources are statically allocated for all jobs and the gradients cannot be aggregated on the host. For ATP and A^{2}TP, the lack of aggregators lead to severe link bandwidth congestion. Thus, their performances also degrade as the aggregator resources decrease. Ring-Allreduce is not affected from the size of switch memory. If the switch memory are extremely scarce, it might be better to use Ring-Allreduce for a part of jobs.

8 Discussion

Fairness Issues. (1) How to ensure minimum fairness for a straggler job? According to A^{2}TP's design in Section 4.3.2, in the worst case, the jobs with severe straggler will relinquish almost all of the aggregators to the job with non-straggler. However, they can still send gradients to PS for aggregation, bypassing the aggregator. The bandwidth allocation at the bottleneck link to PS will eventually converge to fairness. Therefore, the job experiencing severe straggler will not be starved, ensuring minimum fairness.

(2) How to achieve fair allocation of aggregators for multiple jobs? Under A^{2}TP, for workers with the same straggler degree, they will follow the same resource allocation rules of Additive Increase and Multiplicative Decrease (AIMD), so they will eventually converge to fairness. Meanwhile, if the enterprise is more concerned about job's fairness than efficiency, the straggling degree could be fixed to the same to achieve fair resource allocation among multiple jobs.

Comparison with Ring-Allreduce. In Ring-Allreduce algorithm, the amount of training data that each worker sends and receives is $\frac{4(N-1)M}{N}$ [3], where N and M are the number of workers and the total number of bytes to be aggregated, respectively. Under A^{2}TP, the amount of data that each worker sends and receives is $2M$ and the amount of data that PS sends and receives is also $2M$. It indicates that Ring-Allreduce needs more time to communication. If the aggregator resource suffers from congestion, A^{2}TP employs aggregator congestion control to improve the aggregation efficiency, which still provides low communication overhead. Figure 13 shows that A^{2}TP achieves lower average iteration time than Ring-Allreduce under large-scale scenarios. In the extreme case, the lack of aggregator resources leads to severe link bandwidth congestion, it might be better to use Ring-Allreduce for a part of jobs.

Small payload and floating-point calculation. The INA solutions such as SwitchML, ATP, and A^{2}TP are implemented on Intel Tofino switch, which can only support up to 256B of packet payload for aggregation due to limitation of computation flexibility. SwitchML provides a optional manner to expand payload size to 1024B by recirculating packets through four switch piplines. However, it needs to place extra switch ports into loopback mode. Besides, since the switch dataplanes cannot support float-point calculations, such INA solutions adopt floating-point value quantization to converts the floating-point values into 32bit integers at the end-host, incurring small aggregation error. Fortunately, PANAMA [5] uses FPGA to support the floating-point calculations. Above issues are caused by the insufficient capability of programmable switch. Using next generation of programmable switch or FPGA may address these issues.

Implementation with high performance solutions. For RDMA, RDMA Reliable Connected (RC) mode provides flow control and reliable delivery in the RNIC for point-to-point communication. However, A^{2}TP is designed for in-network aggregation, breaking the point-to-point semantics. Thus, RDMA RC is not fit for A^{2}TP, and we implement a new aggregator-aware congestion control algorithm by using UDP-like Raw Ethernet. It is beneficial for high throughput and low latency of RDMA. Since we implement flow control, reliable delivery, and aggregation operations in userspace by using CPU, GPU memory is not yet handled by A^{2}TP. Thus, A^{2}TP is not beneficial for GPUDirect RDMA yet. We leave this work in the future. For higher performance GPU such as V100, it achieves a shorter computation time, and the communication time becomes a larger part in each iteration. Therefore, using in-network aggregation will obtain more benefit due to reducing the traffic volume of communication phase. The resource requirement of in-network aggregation does not increase, since the required switch memory resources are related to the number of jobs and the size of the bandwidth-delay product (BDP).

9 Related Work

Many communication optimization schemes have been proposed to accelerate DML training by reducing the traffic transferred in the network.

The first category is to reduce the traffic volume by the in-network aggregation at the switches. DAIET [31] proposes a concept design of INA, which aggregates gradients at the switch to compress traffic volume on the whole path. Both Sharp [32] and Sharpv2 [33] propose INA protocols for Infiniband. Sharp maps the aggregation tree to the switches in the network and aggregates data that needs to be aggregated in each layer according to the aggregation tree. Although Sharp is able to reduce the transmitted data in the network, it cannot tolerate packet loss to provide reliable transmission and is also unable to be deployed on the Ethernet.

To solve the above problems, SwitchML [4] implements parameter aggregation with a static and fair resource allocation mechanism on programmable switches in Ethernet network. Besides, SwitchML proposes a packet loss recovery mechanism for INA transmission to ensure the reliability of aggregation. PANAMA [5] designs a lightweight FPGA-based in-network aggregation module, which supports floating-point operations and achieves linear aggregation speed without affecting the training accuracy. Meanwhile, PANAMA designs a congestion control mechanism for in-network aggregation transmission, providing better in-network aggregation service than SwitchML under the large-scale shared cluster scenarios. However, the static allocation of aggregator resources in SwitchML and PANAMA may result in aggregator under-utilization. To solve this problem, ATP [6] dynamically and fully utilizes aggregator resources. ATP sends gradients to aggregators with best effort for aggregation, and supports gradients to be aggregated at the PS when aggregator resources are insufficient.

Besides, there are several other works to speed up the transfer of gradient at the end host. For example, OmniReduce [34] considers the sparsity of gradients, which only sends the non-zero gradients to reduce the traffic volume. OmniReduce creates indexes for non-zero gradients, and the aggregator records the non-zero gradient index of all hosts to confirm the aggregation progress of the current aggregation gradient. With the cooperation between the host and the aggregator, the host only needs to transmit non-zero gradients during the aggregation process, effectively improving the throughput. Moreover, P3 [35] greatly reduces the idle waiting time of GPU through overlapping computation and communication. ByteScheduler [36] further optimizes the overlap of communication and computation by dividing the gradient into fine granularity. Meanwhile, several new communication patterns are proposed. For example, BytePS [3] combines All-Reduce and PS communication patterns to fully utilize available bandwidth. BlueConnect [37] proposes a topology-aware adaptive communication mechanism, which adjusts the communication pattern in accordance with network conditions.

Compared with the above works, A^{2}TP focuses on the congestion control mechanism and allocation of aggregator resources in existing INA services. Through the decoupled congestion control and straggling estimation mechanism, aggregator resources are efficiently allocated among multiple jobs to improve aggregation efficiency and reduce the model training time.

10 Conclusion

In this paper, we propose aggregator-aware aggregation transmission protocol A^{2}TP, which decouples the congestion control of in-switch aggregators and link bandwidth, and allocates aggregators according to the straggling degree for multiple jobs to alleviate the impact of straggler job on aggregation process. We implement A^{2}TP by using a P4-programmable switch and a kernel bypass protocol stack at the end host. We evaluate A^{2}TP in the hardware testbed with the real-world benchmark training models. The results show that A^{2}TP speeds up the training time by up to 66% than existing INA protocols. A^{2}TP's source code is publicly available at https://github.com/CSU-NetLab/A2TP-Eurosys2023.

Acknowledgments

We would like to thank our shepherd Anurag Khandelwal and the anonymous reviewers for their insightful comments. This work was supported by the National Natural Science Foundation of China (62132022), Key Research and Development Program of Hunan (2022WK2005), and Natural Science Foundation of Hunan Province, China (2021JJ30867).

References

[1] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[2] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[3] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters. In *Proc. USENIX OSDI*, pages 463–479, 2020.

[4] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtarik. Scaling distributed machine learning with in-network aggregation. In *Proc. USENIX NSDI*, pages 785–808, 2021.

[5] Nadeen Gebara, Manya Ghobadi, and Paolo Costa. In-network aggregation for shared machine learning clusters. In *Proc. MLSys*, pages 829–844, 2021.

[6] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael Swift. Atp: In-network aggregation for multi-tenant learning. In *Proc. USENIX NSDI*, pages 741–761, 2021.

[7] Barefoot networks. https://barefootnetworks.com/products/brief-tofino/.

[8] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant gpu clusters for dnn training workloads. In *Proc. USENIX ATC*, pages 947–960, 2019.

[9] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *Proc. USENIX NSDI*, pages 311–324, 2016.

[10] Zhuolong Yu, Chuheng Hu, Jingfeng Wu, Xiao Sun, Vladimir Braverman, Mosharaf Chowdhury, Zhenhua Liu, and Xin Jin. Programmable packet scheduling with a single queue. In *Proc. ACM SIGCOMM*, pages 179–193, 2021.

[11] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27:19–27, 2014.

[12] Chloe LeGendre, Wan-Chun Ma, Graham Fyffe, John Flynn, Laurent Charbonnel, Jay Busch, and Paul Debevec. Deeplight: Learning illumination for unconstrained mobile mixed reality. In *Proc. IEEE CVPR*, pages 5918–5928, 2019.

[13] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *Proc. USENIX NSDI*, pages 185–198, 2013.

[14] Mingran Yang, Alex Baban, Valery Kugel, Jeff Libby, Scott Mackie, Swamy Sadashivaiah Renu Kananda, Chang-Hong Wu, and Manya Ghobadi. Using trio: juniper networks' programmable chipset-for emerging in-network applications. In *Proc.ACM SIGCOMM*, pages 633–648, 2022.

[15] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Fan Yang, Lidong Zhou, Mao Yang, Francis CM Lau, Yuqi Wang, Yifan Xiong, et al. Hived: Sharing a gpu cluster for deep learning with guarantees. In *Proc. USENIX OSDI*, pages 515–532, 2020.

[16] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *Proc. USENIX OSDI*, pages 595–610, 2018.

[17] Weiyan Wang, Cengguang Zhang, Liu Yang, Kai Chen, and Kun Tan. Addressing network bottlenecks with divide-and-shuffle synchronization for distributed dnn training. In *Proc. IEEE INFOCOM*, pages 320–329, 2022.

[18] Liang Luo, Peter West, Arvind Krishnamurthy, Luis Ceze, and Jacob Nelson. Plink: Discovering and exploiting datacenter network locality for efficient cloud-based distributed training. In *Proc. MLSys*, 2020.

[19] Nvidia, nvidia dgx-1, 2017. http://www.nvidia.com/dgx-1.

[20] Cheng Wang, Bhuvan Urgaonkar, Neda Nasiriani, and George Kesidis. Using burstable instances in the public cloud: Why, when and how? In *Proc. ACM Measurement and Analysis of Computing Systems*, 2017.

[21] Riza O Suminto, Cesar A Stuardo, Alexandra Clark, Huan Ke, Tanakorn Leesatapornwongsa, Bo Fu, Daniar H Kurniawan, Vincentius Martin, Maheswara Rao G Uma, and Haryadi S Gunawi. Pbse: A robust path-based speculative execution for degraded-network tail tolerance in data-parallel frameworks. In *Proc. SoCC*, pages 295–308, 2017.

[22] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proc. ACM SIGCOMM*, pages 63–74, 2010.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE CVPR*, pages 770–778, 2016.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84–90, 2012.

[26] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[28] Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. Multi30k: Multilingual english-german image descriptions. In *Proc. Workshop on Vision and Language*, pages 70–74, 2016.

[29] Liang Luo, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. Parameter hub: a rack-scale parameter server for distributed deep neural network training. In *Proc. ACM SoCC*, pages 41–54, 2018.

[30] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[31] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. In-network computation is a dumb idea whose time has come. In *Proc. ACM HotNets*, pages 150–156, 2017.

[32] Richard L Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, et al. Scalable hierarchical aggregation protocol (sharp): a hardware architecture for efficient data reduction. In *Proc. IEEE COMHPC)*, pages 1–10, 2016.

[33] Richard L Graham, Lion Levi, Devendar Burredy, Gil Bloch, Gilad Shainer, David Cho, George Elias, Daniel Klein, Joshua Ladd, Ophir Maor, et al. Scalable hierarchical aggregation and reduction protocol (sharp) tm streaming-aggregation hardware design and evaluation. In *Proc. HiPC*, pages 41–59, 2020.

[34] Jiawei Fei, Chen-Yu Ho, Atal N Sahu, Marco Canini, and Amedeo Sapio. Efficient sparse collective communication and its application to accelerate distributed deep learning. In *Proc. ACM SIGCOMM*, pages 676–691, 2021.

[35] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based parameter propagation for distributed dnn training. In *Proc. MLSys*, pages 132–145, 2019.

[36] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed dnn training acceleration. In *Proc. ACM SOSP*, pages 16–29, 2019.

[37] Minsik Cho, Ulrich Finkler, David Kung, and Hillery Hunter. Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. In *Proc. MLSys*, volume 1, pages 241–251, 2019.