Future Generation Computer Systems 174 (2026) 107983

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Approximate Gradient Synchronization With Adaptive Quantized Gradient Broadcast☆

Shouxi Luo [a,b,c,*], Xue Liu [a], Ke Li [a], Huanlai Xing [a], Xu Zhang [d]

[a] School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu, 611756, China
[b] Manufacturing Industry Chains Collaboration and Information Support Technology Key Laboratory of Sichuan Province, Southwest Jiaotong University, Chengdu, 611756, China
[c] Tangshan Institute, Southwest Jiaotong University, Tangshan, 063000, China
[d] University of Leeds, Leeds, LS2 9JT, United Kingdom

ARTICLE INFO

Keywords:
Distributed deep learning
Communication optimization
Computation-communication overlap
Quantization

ABSTRACT

Given the layered training workflow of deep neural networks, recent advances have shown that, by splitting gradients into blocks and rearranging their transmission, distributed deep learning (DDL) workers can overlap parts of the communication with computation to hide the overhead of model synchronization. However, not all communication can be masked perfectly (e.g., that of the first-layer gradients). A promising solution is to transmit quantized gradients instead of raw values to eliminate the communication bottleneck further. In this paper, we propose AQGB, Adaptive Quantized Gradient Broadcast, to accelerate the convergence of data-parallel distributed training through designing efficient multi-level quantization and flexible quantization ratio control. Distinguished from existing fixed-quantization schemes, AQGB can adjust the level of quantization respecting the network state and the training progress to maximize the computation-communication overlap (CCO), which is quantified by a novel metric ROW (the Ratio of Overlap time to Wait time). Compared with no-quantization and 4bit-fixed QSGD quantization, AQGB could accelerate the convergence speed of training (regarding the time to converge) by about 3.15× and 1.24×, respectively.

1. Introduction

To guarantee the convergence of training, workers in data-parallel distributed deep learning (DDL) generally have to synchronize their local results (e.g., gradients) before iterating to the next round of training [1–7]. Recent studies have shown that the communication triggered for the synchronization of model parameters could dominate the entire training, becoming the system bottleneck [8,9]. Such an issue is getting more serious as large models are getting popular [2,3,10]. To deal with this, numerous optimization designs including *data compression* [5], *tensor fusion* [6], and *communication scheduling* [4,7], are proposed, to reduce the time that training workers are blocked by communication, by reducing the traffic volume and/or by overlapping computation with independent communication [4,5,11]. Indeed, as orthogonal, these schemes can be employed jointly. For instance, using gradient quantization to enhance communication scheduling is a promising solution: as deep neural networks (DNNs) are trained layer-by-layer, by reordering the tensor transmissions for different layers to overlap communication with computation, workers could hide some of the communication overheads [4,7]; and for communication that cannot be masked, by transmitting quantized-yet-error-controlled gradients rather than their original values, workers can further reduce the involved traffic load along with the hanging time, thus accelerating the training iteration [12].

However, as a lossy compression technique, gradient quantization has the possible cost of reduced model accuracy or increased rounds to convergence [5] (see Fig. 2). Moreover, in large-scale shared clusters, because various distributed applications are likely to co-locate [13,14], the available bandwidth a transfer could use is time-varying. Accordingly, a perfect quantization scheme should have the ability to adapt its level of compression respecting the network dynamics, given that compression schemes with better quality might lead to a

☆ This work was supported in part by NSFC, China under Grant 62002300, in part by NSFSC, China under Grant 2025ZNSFSC0489, in part by the Hebei Natural Science Foundation, China under Grant F2025525008, and in part by the Fundamental Research Funds for the Central Universities, China under Grant 2682024ZTPY050. A brief description of the preliminary design of this paper was presented in the Poster session of APNet 2022 as a 2-page extended abstract DOI:10.1145/3542637.3543708.

* Corresponding author.
E-mail address: sxluo@swjtu.edu.cn (S. Luo).

https://doi.org/10.1016/j.future.2025.107983
Received 7 February 2025; Received in revised form 11 June 2025; Accepted 18 June 2025
Available online 1 July 2025

better model quality or faster convergence [5,15–17]. Unfortunately, to the best of our knowledge, none of the existing schemes have supported this, as they all adopt fixed and inflexible quantization designs [5], resulting in inefficient performance in terms of both the computation-communication overlap (**CCO**) and the utilization of network bandwidth. More specifically, when there is sufficient network bandwidth, a strict fixed quantization setting would suffer from low bandwidth usage and high quantization loss; on the contrary, when the available network bandwidth is low, a mild fixed quantization configuration would lead to very long communication times, which might hang the dependent training computation.

To fill the gap, in this paper, we propose AQGB (Adaptive Quantized Gradient Broadcast),[1] a novel approximate gradient synchronization scheme for DDL, embodying the idea of dynamically adjusting the quantization level of gradients respecting both the training progress and network status. Generally, in DDL, it could take up to thousands of rounds of iterative training for a model to converge; and during this progress, gradients are produced (by the backward-propagation), then consumed (by the forward-propagation) in layer-wise, or more refined, in block-wise manners [4]. By profiling several rounds of training, it is possible to estimate the (soft) deadline for the transmission of each gradient, respecting the goal of maximizing the CCO. Based on the profiled deadlines and updated network status, AQGB could achieve "perfect" overlapping and network utilization in a dynamic system.

Despite being attractive, making the above idea come true is non-trivial, as the following design challenges must be addressed. First of all, gradient compression (e.g., quantization) could reduce the traffic volume; however, as we will show in this paper, it might also reduce the overlap between computation and communication (i.e., over-killing); thus, to avoid this problem, we need a suitable metric to evaluate and act as the optimization goal (i.e., **C1**). Secondly, to reduce the wait/blocked time while maximizing the CCO, we need a scheme to dynamically adjust the level of quantization, respecting the training progress and network status (i.e., **C2**). Last but not least, to support adaptive gradient quantization, we need a novel scheme supporting multi-level compression with low encoding/decoding overheads (i.e., **C3**).

To address **C1**, we propose the novel metric of ROW, which could capture both the wait time of workers and the achieved CCO, and use it as the optimization goal of AQGB. Then, based on ROW, we design a progress-aware quantization ratio control algorithm for AQGB, enabling training workers to adjust their level of quantization respecting the training progress and network state, in a best-effect manner, thus addressing **C2**. To support hierarchical gradient quantization and deal with **C3**, we further design a flexible and efficient encode/decode scheme of TFP, based on the truncation of floating-point numbers for AQGB. It is worth mentioning that, as a generic approximate gradient synchronization framework, AQGB can also work with other multi-level quantization schemes beyond TFP; and we leave this as future work.

Extensive performance studies confirm that, for a targeted compression ratio, the impact of our proposed TFP on the training convergence is quite similar to that of the well-known QSGD quantization scheme [18], with much simple truncation-based encode/decode designs; and by dynamically adjusting the compression ratio of TFP-coded gradients respecting the network status and training progress to maximize CCO, AQGB could make more efficient use of the network to accelerate the convergence of the distributed training. For example, in our tests, compared with no quantization and 4 bit fixed QSGD gradient quantization, AQGB could accelerate the convergence speed of training (in terms of the time to converge) by about 3.15× and 1.24×, respectively.[2]

[1] AQGB is designed to be decoupled from the transport protocols and sits at the application layer. This allows it to work with any existing reliable communication protocol, depending on what the underlying network provides.

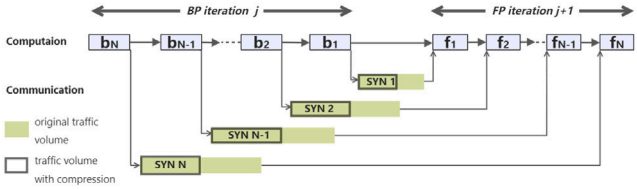


Fig. 1. The workflow of workers training a DNN model with data-parallel SGD algorithms.

In summary, this paper mainly makes five contributions.

- An analysis of the demands and metric of communication optimization for data-parallel DDL (Section 2);
- ROW (**R**atio of the **O**verlap time to **W**ait time), a metric capturing both the wait time of workers and CCO (Section 3);
- AQGB, an approximate-yet-consistent gradient synchronization framework built upon adaptive quantized gradient broadcast, to optimize the ROW for DDL (Section 4.2);
- A progress-aware quantization ratio control algorithm that could adjust the level of quantization respecting the training progress and network state (Section 4.3); and
- TFP, a flexible and efficient encode/decode scheme based on the Truncation of Floating-Point numbers, that supports hierarchical gradient quantization (Section 4.4).

In the rest of the paper, we first introduce the background and motivations in Section 2, then describe the definition of ROW in Section 3. After that, we look into the design details of AQGB in Section 4. Performance studies and related work are presented in Sections 5 and 6, respectively; and finally, we conclude the paper in Section 7.

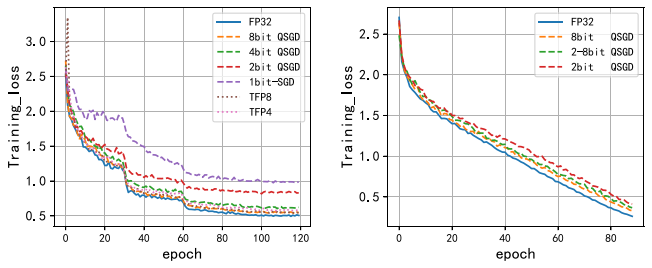
2. Background and motivation

As the background, we first overview the workflow of data-parallel DDL (Section 2.1). Based on this, we then analyze why CCO is the right metric for the involved communication optimization, which motivates the design of AQGB (Section 2.2). Finally, we discuss the limits of existing solutions (Section 2.3).

2.1. Data-parallel distributed deep learning

Nowadays, DNN models are generally trained with the well-known algorithm of Stochastic Gradient Descent (SGD) or its variants with iterative designs. In each round, a batch of training data samples will be selected. As Fig. 1 shows, with these samples, a training algorithm first conducts a *forward propagation* (FP) to obtain the loss values; based on which, a *backward propagation* (BP) is then processed to update the values of model parameters. Thanks to the specific structure of DNNs, both the forward and backward progress are conducted in layer-wise, or even more fine-grained, block-wise manners [4]. In data-parallel distributed training, the same model is updated by a group of workers, each of which only holds a part of the entire training data. To guarantee the convergence of distributed training, workers would synchronize their locally updated models or gradients periodically during the training. Obviously, following such a design, a worker can start the synchronization of a parameter, once it is generated during the backward propagation; and the worker could not conduct

[2] Let t_A and t_B be the time workers need to complete a given round of training under the scheme of A and B, respectively; then, the speedup of A over B is defined by $\frac{t_B}{t_A}$.



(a) ResNet50 w/o EF (b) ResNeXt101 w/ EF

Fig. 2. Gradient quantization could lead to slower convergence. Consider 4-bit quantization and 8-bit quantization as examples, our proposed TFP would achieve comparable performances with QSGD in terms of the impacts on the training loss.

the related forward propagation computation in the next round until the synchronized model (or gradient) value is obtained [4].

Abundant recent studies [4–6,19] have shown that when training large models over a massive dataset with a large number of workers, the time it takes for workers to conduct the synchronization would dominate the entire train; thus, optimizing the communication involved becomes critical for the performance optimization of large-scale DDL training.

2.2. The right metric for communication optimization

In practice, workers in DDL can execute the model synchronizations with various communication designs, including parameter server, ring- or tree- based allreduce, peer-to-peer direct delivery, and random gossip [20–23]. To relieve the bottleneck effects of the involved communication, a large amount of model or gradient compression methods have been proposed. According to [5], they can be classified into four types, namely, *quantization*, *sparsification*, *hybrid*, and *low-rank*, respectively. At a high level, all these schemes target the goal of reducing the impacts of communication bottlenecks in synchronization to make DDL more efficient. However, at the low level, as compression-based solutions, they mainly pursue the direct objective of reducing the volume of involved traffic, yielding a gap from the original goal. Such a mismatching might lead to a loss of performance.

More specifically, as Fig. 1 shows, DNN models are trained in a layer-wise manner; and the effects of communication bottleneck can be reduced by pipelining the computation with independent communication. For example, for DDL, workers can make a layer's FP and BP computation overlap with the communication triggered by a deeper layer, using pipeline and layer-aware flow scheduling techniques [4,6]. By reducing the traffic volume involved in model synchronization, compression techniques are able to cut down the time cost of communication. However, they might be over-killing—Since the synchronization of some layers' parameters might not be the communication bottleneck, hence compressing them to reduce the traffic volume might not improve the CCO. Moreover, recent studies have shown theoretically and empirically that, *in general, for a compression scheme, increasing the compression quality (e.g., sending more data to reduce the variance) for the synchronization could lead to a better model accuracy or faster convergence* [5,15–17].

As Fig. 2 shows, we have witnessed such phenomena in experiments (where TFP is the quantization scheme we proposed in this paper, detailed in Section 4.4). Fig. 2(a) demonstrates the declines of training the ResNet50 model upon CIFAR10, under different levels of QSGD-based gradient quantization settings. Here, the workload is distributed among 4 workers with mini-batchsize = 128; on each worker, except for the case of 1bit-SGD, the SGD optimizer is used without enabling Error-Feedback (EF); the learning rate is set to 0.1 initially and would be decayed 10 every 30 epochs. It is obvious that, for the EF-disabled training of ResNet50, the higher level of quantization significantly leads to a slower convergence speed; and even worse, because of the lossy gradient quantization, the model would finally converge to a much worse accuracy. In practice, for some models, the side effect of lossy gradient quantization can be partially relieved by the design of EF. Unfortunately, due to its high extra memory occupy, the EF mechanism might be disabled when training very large models [24]. Moreover, as recent studies show [25], there are some types of models like ResNeXt101, GNMT, Mask RCNN, and BERT, that are very sensitive to the noise in gradient quantization and EF designs are unable to fix. For instance, as Fig. 2(b) shows, even with EF, the convergence speed of training ResNeXt101 upon four workers using the Adam optimizer with the learning rate of 1e−4, can be slowed down by the quantization of gradients. Here, `2-8bit` refers to the setting of randomly compressing each gradient value with either 2 bit QSGD or 8 bit QSGD, with the equal probability. Thus, existing compression schemes are far from optimal due to their misleading low-level optimization goals.

Despite some recently proposed schemes like DC2 [19], AQG [26], and AC-SGD [27] having the ability to adjust the compression ratio dynamically, we argue that they suffer from a similar problem as well since they are essentially agnostic to the opportunity of CCO in DDL, by design.

In a nutshell, for DDL tasks, directly maximizing the CCO, rather than other alternative metrics like the reduced traffic volume, is a better optimization goal.

2.3. Limits of existing solutions

We are not the first who directly targets the goal of maximizing the CCO. For example, ooo (out-of-order) [28] tries to maximize the overlap by reordering the backprop progress; ByteScheduler [4,29] use the design of rescheduling the high-level transmission order of low-level flow/packet priorities. However, since only using scheduling or reordering techniques, which would not change the total traffic volume, they are with limited abilities and are unable to deal with the case where the available bandwidth will change in dynamic. Indeed, by compressing the synchronization communication for DDL in a progress-aware manner, training workers can not only optimize the CCO but also be able to keep the same overlap to mask the possible computation or communication stragglers, by adjusting the compression ratio adaptively.

To achieve the above goal, the foundation is to quantify the CCO explicitly. Despite several recent works having tried [30,31], as we will show in the next section, their proposed metrics fail to capture the adaptive compressibility of the involved transfers, leaving room for improvement.

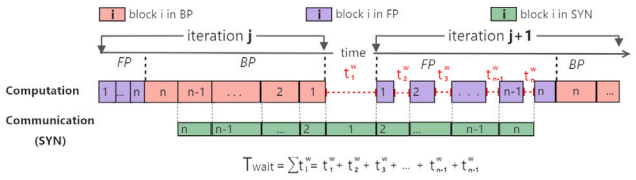
3. Quantifying the overlap

Before presenting the detail of our proposed AQGB, in this section, we look into the foundational problem of how to formally quantify the CCO for DDL. We first discuss the drawbacks of existing schemes, T_{wait} [31] and OSF [30] (Section 3.1), then propose our answer, ROW, a better metric defined as the **R**atio of the **O**verlap time to **W**ait time, for the problem (Section 3.2).

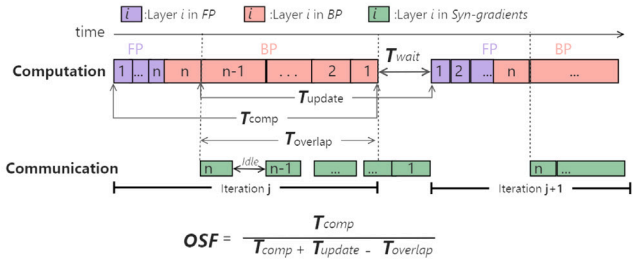
3.1. Drawbacks of T_{wait} and OSF

The overlap between the computation and communication can be partially quantified by T_{wait} and OSF.

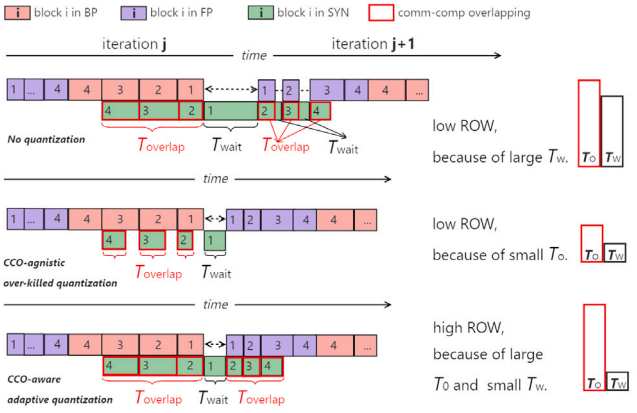
Recall that, in DDL, a worker could not conduct FP on a layer before the synchronization of the corresponding parameters is completed; thus a good CCO optimization should greatly reduce the time that each work has to wait for the synchronization to complete. Motivated by this, the work of [31] uses T_{wait}, the total wait time each worker has encountered in a round of training (e.g., $\sum_i t_i^w$ in Fig. 3(a)) as the



(a) An example showcasing the definition of T_{wait}.



(b) An example showcasing the definition of OSF.



(c) Examples showcasing the definition of ROW and its advantages over T_{wait} and OSF. Here, T_o and T_w are the short names of $T_{overlap}$ and T_{wait}, respectively.

Fig. 3. Examples showcasing the definitions of T_{wait} [31], OSF [30], and our proposed ROW, along with why ROW is better than the other two. Note that, the T_{update} used by OSF is generally larger than the actual communication time, as it also includes the idle time slots caused by slow BP. Also, $T_{overlap}$ is defined as the overlap between T_{comp} and T_{update}; while, the $T_{overlap}$ in ROW is defined as the overlap between T_{comp} and T_{comm}, which is more accurate.

metric to minimize. However, as an implicit metric, it is unable to capture the CCO during the BP computation; thus, like the case of current CCO-agnostic compression techniques, a scheme might be over-killing on reducing the traffic volume when only trying to minimize T_{wait} (e.g., Fig. 3(c)).

Besides T_{wait}, more recently, the work of [30] proposes OSF, i.e., *Scaling Factor considering Overlap*, defined as $\frac{T_{comp}}{T_{comp}+T_{update}-T_{overlap}}$, to formulate the scalability of data-parallel DDL tasks, by taking the CCO into consideration. As Fig. 3(b) shows, given a round of training, T_{comp} is the total computation time; T_{update} is the duration starting from the beginning of model synchronization to its end, which is generally larger than the actual communication time, as it also includes the idle time slots caused by slow BP computation; and $T_{overlap}$ is computation time slots that overlap with T_{update}. Obviously, such a design overlooks the fact that parts of the synchronization communication can overlap with the next round of FP processing as well, as workers could not start the FP until the entire synchronization completes. Thus, OSF falls short. Indeed, in such a setting (i.e., no communication overlaps with the FP computation), there would be no wait time involved in each FP; then, we would have $T_{comp} + T_{update} - T_{overlap} = T_{comp} + T_{wait}$ and $OSF = \frac{T_{comp}}{T_{comp}+T_{wait}}$. That is to say, given a training task, maximizing the OSF is equivalent to minimizing the T_{wait}. Thus, OSF suffers from the same problems faced by T_{wait}.

3.2. Our proposed ROW

According to the workflow of DDL shown in Fig. 1, the straightforward design is to directly use the duration of the overlap time between the computation and communication as the quantification of CCO, and as the metric to optimize. However, like T_{wait}, such a definition would work fine for communication optimization designs that would not change the size of traffic volume (e.g., flow scheduling alone). Once compression techniques are employed, it is unable to capture the possibility that, the wait time before the FP process of the first layer, can be reduced by compression techniques as well.

Taking all the above considerations into account, in this paper, we propose ROW, the *Ratio of Overlap time to Wait time*, to quantify the overlap, as Eq. (1) shows. Different from OSF (Fig. 3(b)), the $T_{overlap}$ used in ROW (Fig. 3(c)) is exactly the total overlap time between the T_{comp} and T_{comm} (rather than T_{update}). Obviously, by pursuing the objective of maximizing ROW, a communication optimization scheme would be enforced to only reduce the bottleneck. As examples in Fig. 3 show, ROW is more powerful than T_{wait} and OSF, as it supports compression-based optimizations and captures whether the compression is over-killing or not.

$$\text{ROW}\ (of\ a\ worker) := \frac{T_{overlap}}{T_{wait}} \tag{1}$$

In practice, both computation and communication stragglers are prone to occur during training [32]. Moreover, when adaptive compression techniques are employed, the traffic volume would not be fixed. Thus, for a given distributed training task, workers are likely to have various $T_{overlap}$s and T_{wait}s, and their values might change with time. Let $T^i_{overlap}$ and T^i_{wait} be the current overlap time and wait time experienced by worker i. We define the ROW of the entire cluster as the ratio of all workers' minimum overlap time to their maximum wait time, as Eq. (2) specifies.

$$\text{ROW}\ (of\ a\ group\ of\ m\ workers) := \frac{\min_{i=1}^{m} T^i_{overlap}}{\max_{i=1}^{m} T^i_{wait}} \tag{2}$$

4. AQGB

Based on ROW, we propose AQGB, an approximate gradient synchronization scheme built upon Adaptive Quantized Gradient Broadcast, for data-parallel DDL. Given dynamic network environments, AQGB performs adaptive quantization in a best-effort manner, using a suite of simple yet effective scheduling principles. In short, AQGB implements its adaptive quantization control at the application layer and adjusts the level of quantization with respect to the observed bandwidth, the un-delivered volume, and the remaining time to the desired completion time dynamically. Such a design enables AQGB to deal with variability like congestion and heterogeneous computing power, in distributed training environments appropriately. For example, when network congestion occurs, the un-delivered volume would be larger than expected, making the sender increase its quantization level; when a worker becomes a straggler and is unable to complete its delivery before the expected time, it would increase its quantization level accordingly. While for other workers, they would have sufficient time to complete the delivery and thus would decrease its quantization level. Regarding potential node and link failures, AQGB neither specifically optimizes for this aspect nor introduces additional vulnerabilities. Consequently, existing fault-tolerant training mechanisms such as checkpoint-based recovery [33] remain fully applicable and can be seamlessly integrated with our approach.

In the following, we first describe the design insights of AQGB in Section 4.1, then overview its main design in Section 4.2; after that, we present the principles AQGB uses to adjust quantization ratios respecting stragglers (Section 4.3), then present our flexible encode/decode scheme that enables AQGB to support multi-level quantization (Section 4.4), in detail.

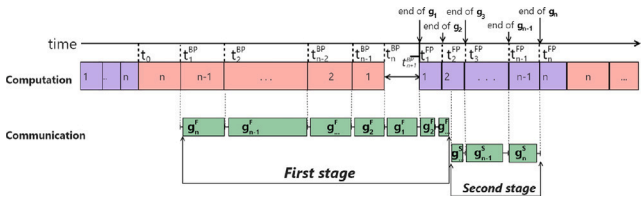


Fig. 4. The two-stage gradient broadcasting of AQGB.

4.1. Design insights

As higher compression quality generally leads to faster training convergence or better model accuracy, an attractive design is exploring the trade-offs between quantization levels and model accuracy, to accelerate the total time cost of model training. However, we argue that it is hard, if not impossible, to explore such trade-offs in our context, due to two reasons. Firstly, as is known, the development of new DNNs is time-costly, following a trial-and-error way [34]. In the early stage of development, the relationship between the quantitation level and the training accuracy is unknown and unpredictable. Even worse, there might not exist such trade-offs. Secondly, even if such trade-offs are observed, as reported by various recent studies [22,34,35], the convergence behavior of distributed training jobs is heavily determined by a lot of factors such as the quality of the training datasets, the hyperparameter settings like the batch size, the training algorithm, learning rate, etc., jointly. Consequently, the relationship between the quantization level and the model's convergence behavior is job specific. For example, as our tests explained in Section 2.2 show, (i) compared to ResNet50, ResNeXt101 is more sensitive to the introduced compression errors; and (ii) for both models, enabling error compensation would greatly relieve the side effects of quantization.

$$Q_{ratio} := \frac{\text{Data size after quantization}}{\text{Original data size}} \tag{3}$$

As specified in Eq (3), in this paper, we define the quantization ratio as the ratio of the data size after quantization to the original data size. That is to say, a higher quantization ratio value indicates a milder level of quantization. To provide a generic communication optimization design that benefits various distributed training jobs, AQGB performs adaptive quantization in a best-effort manner: it guarantees that the selected quantization level would never be smaller than a user-specified value and tries to decrease the quantization level as much as possible, providing that the waiting time of computation caused by communication would not be enlarged.

4.2. Solution overview

Currently, AQGB is specialized in optimizing the model synchronization for peer-to-peer DDL with adaptive quantization designs. In AQGB, a worker would immediately broadcast its quantized gradients to all other workers, using supported reliable transport mechanisms like MPI_Bcast, TCP, and reliable multicast transport protocol(s); then, by aggregating the received gradients (including these generated by itself) and applying the results to its local model, each worker obtains the newly globally updated model and moves to the next round of training [12]. At a high level, by profiling the patterns of how gradients are generated and consumed locally, workers in AQGB could estimate the best completion times (i.e., soft deadlines) for the broadcast of its local gradients. Then, at the low level, during the broadcasting, based on both the un-delivered gradients and the expected soft deadline, each worker dynamically changes the level of quantization to react to the presence of stragglers and bandwidth dynamics.

To simplify the control of quantization while maximizing the entire DDL task's ROW, AQGB splits a worker's execution of gradient broadcasting into two stages, as Fig. 4 shows. Since the order of how BP generates new gradients are the reverse of how FP consumes them, during both stages, gradients are transmitted in the last come first served order.

In the first stage, AQGB ensures that all gradient values would be quantized (i.e., compressed) and then transmitted. As the new gradients of a neural network model are generated in a layer-wise manner (see Fig. 4 for example), to minimize the possible wait time and to maximize the overlap time at the same time, workers in AQGB try to complete the delivery of all the generated gradient values just before the next block of gradient becomes available (for blocks other than the last), or as soon as possible (for the last block), by adapting their quantization levels. If some gradients have not achieved this, workers would treat all un-delivered gradients and the newly generated gradients as a whole, and try to meet the newly (soft) deadline with adaptive quantization again.

Once all gradients are quantized and delivered, the input data needed by FP are ready for all workers. However, during the FP, updated model values are consumed layer by layer; for some layers, there might still be available idle time before they are used. To make efficient usage of the available bandwidth, workers then switch to the second stage, in which, they try to transmit more data to reduce the quantization errors introduced in the first stage, such that the CCO in the FP could be maximized. Hereafter, for ease of description, we call such data *remaining* data. To control the transmission of remaining data, for the remaining quantized data of layer i, a worker would treat the estimated usage time (i.e., the start time of its next round of the FP process) as a deadline to decide the quantization ratio dynamically. However, during the training, because of stragglers, workers might have various estimated FP start times for each layer. To increase CCO, workers would synchronize their estimated FP start times; and for each layer, a worker would use the latest version of the maximum corresponding FP start time to compute the quantization level for the second stage of gradient broadcasting.

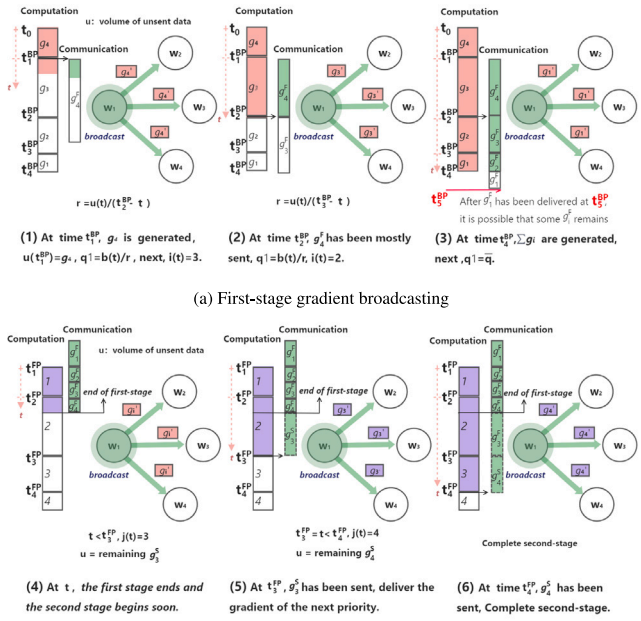
4.3. Adaptive quantization ratio control

Consider that m workers $w_1, \ldots, w_m$ are training a neural network involving n layers; a worker (e.g., w_1) starts its BP at time t_0^{BP}, obtains the $(n+1-i)$th layer's locally trained gradients with the size of g_{n+1-i} at time t_i^{BP}, and launches its next round of FP at time t_1^{FB}. Hereafter, we also use g_i to refer to the corresponding gradient data block. Then, the data of g_i would be quantized and then transmitted with these two stages of broadcasting, which are denoted by g_i^F and g_i^S, respectively. Formally, for a given worker, it starts the first stage of gradient broadcasting when its last layer's gradient block g_n becomes available, then stops the broadcasting once it has delivered all its own g_i^Fs, the quantized values of g_is, to all receivers. After that, the worker starts the second stage of gradient broadcasting, then stops the broadcasting either when all the needed data has been delivered, or when there is no need to transmit them anymore since the worker already starts the FP process on the last layer.

To guarantee that all workers receive exactly the same set of gradients to conduct consistent synchronizations, reliable broadcasting is used for the delivery of gradient blocks in both stages; and a worker would start g_i's FP process, only after getting all the gradient data.

4.3.1. First stage

As described in Section 4.2, to avoid the delivery of the current block becoming the communication bottleneck, the worker wants (1) the delivery of g_i to be complete before g_{i-1} becomes available, and (2) the delivery of the final block g_1, along with all undelivered gradients, to complete as soon as possible. Let $u(t)$ be the original size of the worker's generated-yet-undelivered at time t, and $i(t)$ be the index of the gradient block that will appear next, respectively. Then, to achieve the above goal, the worker's broadcast rate should not be lower than $r = {}^{u(t)}/_{(t_{n+1-i(t)}^{BP}-t)}$. However, the actual broadcast rate this



(a) First-stage gradient broadcasting

(b) Second-stage gradient broadcasting.

Fig. 5. Examples showcasing how the first- and second- stage gradient broadcasting of AQGB works.

worker has achieved is $b(t)$; we use it as the estimation of the available bandwidth in the near future. To eliminate this mismatch, the expected compression ratio, that AQGB should approximate, with quantization, can be estimated as $q_1 = {}^{b(t)}/_{r} = {}^{b(t)(t^{BP}_{n+1-i(t)}-t)}/_{u(t)}$. Suppose that the upper bound of the quantization ratio allowed by the quantization designs/techniques and training tasks is $\bar{q}$, which is a tunable parameter. Then, the actual quantization ratio used would be $\max(\bar{q}, q_1)$. And once the last block of gradients becomes available, to accelerate the completion of the first stage of broadcasting, the worker would try to deliver all un-transmitted gradients with the compression level of $\bar{q}$. Following this, the T_{wait} would be finally reduced.

Fig. 5(a) shows a simple example, where $n = m = 4$. The model involves four layered gradient blocks, whose sizes are g_4, g_3, g_2, and g_1, respectively. In the current round of training, the worker w_1 would generate them at time t_1^{BP}, t_2^{BP}, t_3^{BP}, and t_4^{BP}. Then, at time $t = t_1^{BP}$, the size of w_1's generated-yet-undelivered gradient data is $u(t) = g_4$, whose expected delivery completion time is t_2^{BP}. To complete the broadcast of these gradients before t_2^{BP}, w_1 would try to compress its gradients with the quantization ratio of $q1 = {}^{(t_2^{BP}-t)b(t)}/_{u(t)}$, and the actual quantization ratio is $\max(\bar{q}, q_1)$. During delivery, w_1 would update t, $b(t)$, and $u(t)$ to refresh q_1 accordingly. Once the new gradient blocks like g_3, g_2, and g_1 are generated, both the newly generated block sizes and the next expected deadlines (e.g., t_3^{BP} and t_4^{BP}) would be taken into account.

4.3.2. Second stage

Regarding the second stage of broadcasting, the remaining data of each layer is delivered in the order how they would get used; indeed, not the delivery of all layers from a worker would go through such a phase, depending on whether there is still idle time left before the estimated start time of the corresponding FP process. To make more efficient usage of the network under the existence of straggler, each worker in AQGB periodically broadcasts its locally-estimated start time of each layer's FP to other workers; and based on the received and its own estimated start time, each worker would complete the deadline, i.e., t_i^{FP}, for the delivery of the g_i's *remaining* data. On each worker, as the source of the broadcasting, let $j(t)$ be the index of the gradient block whose second stage of broadcasting the sender would prefer to

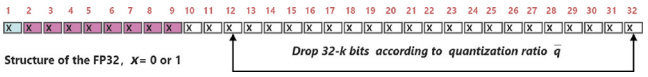


Fig. 6. An example that showcases the usage of AQGB's truncation-based multi-level quantization on FP32.

complete next, and $u(g^S_{j(t)})$ be the remaining volume to be quantized and broadcasted. Then, similar to the case in the first stage of broadcasting, to meet the estimated deadline of $t^{FP}_{j(t)}$ under the estimated available broadcast bandwidth of $b(t)$, the expected quantization ratio of the remaining data of $g_{j(t)}$ is ${}^{b(t)(t^{FP}_{j(t)}-t)}/_{u(g^S_{j(t)})}$.

As an example, Fig. 5(b) shows in detail how w_1 performs the second-stage quantization controls for four gradient blocks. For the ith block, g_i^F denotes its gradient values already delivered in the first stage, and g_i^S indicates its remaining gradients that should be delivered in the second stage. To prevent the FP computation from being blocked by the broadcast communication, w_1 would try to complete the delivery of g_1^S, g_2^S, g_3^S, and g_4^S before t_1^{FP}, t_2^{FP}, t_2^{FP}, and t_2^{FP}, using the quantization ratio of $max(\bar{q}, q_2)$, where q_2 is ${}^{b(t)(t_1^{FP}-t)}/_{u(g_1^S)}$, ${}^{b(t)(t_2^{FP}-t)}/_{u(g_2^S)}$, ${}^{b(t)(t_3^{FP}-t)}/_{u(g_3^S)}$, and ${}^{b(t)(t_4^{FP}-t)}/_{u(g_4^S)}$, respectively.

4.4. Efficient multi-level quantization with TFP

In consideration that gradients in deep learning are generally floating-point numbers like FP16, FP32, and FP64, AQGB designs, TFP (Truncation of Floating Point number), a simple yet efficient multi-level quantization scheme based on the floating-point numbers' specific layouts—i.e., to quantify a gradient into k bits, TFP directly truncates the bit array of the floating-point representations other than the first k bits. Note that, when the targeted level of quantization is small e.g., 4 or 2, before applying truncation AQGB would first convert the gradient into FP16, or FP8, to reduce the introduced errors.

Take the case of quantizing an FP32 gradient into its 12-bit representation as an example. According to the IEEE 754 standard [36], the layout of an FP32 number can be divided into three parts: (1) 1 bit for the sign; (2) 8 bits for the exponent; and (3) 23 bits for the mantissa, as Fig. 6 and Eq. (4) shows. Motivated by this observation and distinguished from existing static quantization schemes, TFP reduces the mantissa bits to achieve quantization on demand, yielding efficient multi-level quantization with the compression ratio of ${}^{k}/_{32}$, where k (≥ 2) is the width of bits that should be kept. Obviously, smaller k values yield higher quantization errors.

$$N(S, E, M) := (-1)^{S} \times 2^{E-127} \times 1.M \tag{4}$$

During the broadcast, quantized gradients are encapsulated as the payload of UDP. To make both encoding and decoding easy to manage, for a given array or stream of gradients made up of s FP32 values, AQGB workers would split them into partitions: by default, each partition would embody p gradient values, yielding the original size of $32p$ bit; but for the last one, it would only hold the remaining $s\%p$ gradients. More specifically, in the first stage of gradient broadcasting, to meet the compression ratio of q_1, for each F32 gradient in a partition, the AQGB worker would directly send its first $L_F = \lceil 32 \max(q_1, \hat{q}) \rceil$ bits, where $\hat{q}$ is with the default value of 0.125; thus, the amount of data to broadcast is roughly reduced from 1 to $\max(q_1, \hat{q})$. For each gradient value, there are $32 - L_F$ remaining bits that the worker could deliver in the second stage of broadcasting. In case the expected compression ratio is q_2, the worker would directly send the first $\lceil q_2(32 - L_F) \rceil$ bits among the remaining. Let l be UDP's limit on the total size of all partitions in a packet. Then, for both stages of broadcasting, workers just greedily pack as many as possible successive partitions to generate a packet, under the constraint of l.

Note that many prior quantization schemes employ *random rounding* to achieve unbiasedness; indeed, TFP supports this as well. Given a

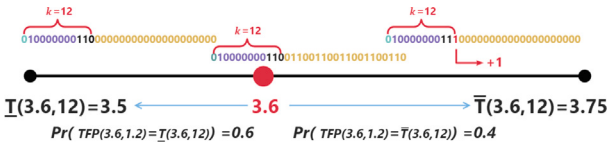


Fig. 7. An illustration of how TFP supports unbiased stochastic quantization using random rounding.

floating-point number x, let $\underline{T}(x,k)$ be its truncated value by only keeping the first k bits, and $\overline{T}(x,k)$ be the truncated value (by only keeping the first k bits again) of the result of adding an "1" to the kth bit of $\underline{T}(x,k)$. Then, the k-bit quantized value of x using TFP with random rounding enabled is computed via Eq. (5). Here, $rand()$ would return a random float uniformly in the range of $[0,1)$ when called. As the example in Fig. 7 shows, for a FP32 number $x = 3.6$, we have $\underline{T}(3.6,12) = 3.5$ and $\overline{T}(3.6,12) = 3.75$; then, TFP$(3.6,12)$ would be 3.5 with the probability of $1 - \frac{3.6-3.5}{3.75-3.5} = 0.6$, or 3.75 otherwise.

$$\text{TFP}(x,k) := \begin{cases} \underline{T}(x,k), & \text{if } rand() \le 1 - \frac{x-\underline{T}(x,k)}{\overline{T}(x,k)-\underline{T}(x,k)} \\ \overline{T}(x,k), & \text{otherwise} \end{cases} \tag{5}$$

In practice, if k is large, the impact of truncation is limited and can easily be masked by techniques like error feedback. To simplify the involved computation, the *random rounding* of TFP can be disabled. However, when k is quite small, *random rounding* would be helpful. In that case, to reduce the introduced quantization errors, TFP would first convert the original x to a more compact float format like FP16, FP8, or even FP4 [37] and then conduct truncation-based quantization.

5. Performance evaluation

To evaluate the performance of AQGB, besides conducting small-scale real training with PyTorch, following recent studies [38–40], we also develop a discrete-event simulator based on the design of ns3 but in Python 3, which could simulate the behavior of a cluster running well-known DDL workloads under various model synchronization schemes including AQGB.

As we have shown in Fig. 2, for a targeted compression ratio e.g., 4 bit, 8 bit, the side effects of our proposed TFP and QSGD on the training convergence speed are very close. Then, we use these observed characteristics profiled from real distributed training to drive the simulation of more complex distributed training. Extensive results indicate that:

- In contrast to T_{wait}, ROW is better since it captures the possible CCO, rather than only the time that workers are blocked by gradient communication.
- AQGB could adjust its level of quantization to cope with dynamic networks, thus making higher usage of the available bandwidth.
- By pursuing the goal of ROW with adaptive gradient quantization, AQGB could make efficient usage of the network as if gradients are not quantized while making the bottleneck T_{wait} time very close to that achieved by fixed-yet-low-bit (e.g., 4 bit) quantization.

5.1. Methodology

5.1.1. Metrics, tools, and baselines

As pointed out by various recent studies [23,34,35], given a distributed model training job, how the trained model converges is jointly determined by abundant factors, ranging from the training dataset to the model structure, to the hardware environment, and to the hyper-parameter settings. Accordingly, to study and highlight the benefits of our proposed schemes in detail, like recent works [23,35], we develop a simulator and mainly employ system-relevant metrics for performance evaluation. More specifically, our simulator could simulate how a group of workers synchronize their locally trained model gradients to drive the DDL upon reliable broadcast.[3] Besides the proposed adaptive quantization AQGB, the simulator also supports other gradient quantization designs. As AQGB is the first work that supports adaptive gradient quantization based on the training progress and network state, to highlight its behaviors in tests, we mainly use the cases with no quantization (i.e., labeled as 32 bit), and 4 bit QSGD-based quantization (i.e., labeled as 4 bit) as baselines. Regarding the metrics, besides our proposed ROW, we also consider T_{wait} (normalized by the average computation time per round of local training), the average utilization of network bandwidth, and the number of DDL training rounds in a given time duration T, with the default value of 200 s, in our tests.

Table 1
ResNet50 settings.

Block ID	Layer	Block size	FP time	BP time
1	conv1–conv4_1	3 m	41.9%	50.5%
2	conv4_2–conv4_6	5.6 m	15.5%	14.2%
3	conv5_1–conv5_1	6 m	16.5%	12.9%
4	conv5_2–fc1000	11 m	26.1%	22.4%
FP:BP training time ratio is about 0.365:0.635				

5.1.2. Cluster and workloads

We consider that m workers are training the well-known ResNet50 model [41] with data-parallel designs. These training workers are assumed to be networked with a top-of-rack switch and with 2.5 Gbps links. By default, $m = 16$. During the training, each worker transmits the gradient blocks generated during backpropagation to other $m-1$ workers by broadcasting. For possible CCO optimization, the model is divided into 4 blocks respecting its structure and number of parameters, as Table 1 specifies. Our basic principle here is to balance the number of parameters in each block, respecting their layer-wise structure. There are also other ways for splitting. As how a model could be split highly depends on its structure, for the optimization of the splitting scheme for different types of models, we leave this as a future direction. For the detailed computation times involved in each block, we use the cycle-accurate CNN accelerator simulator of SCALE-SIM [40] to synthesize the time cost of each step.

In consideration that, during the training, the time needed by a worker to conduct a round of training is not exactly the same because of system noise, we randomly scale the time values generated by SCALE-SIM [40] to synthesize their time costs. In tests, we consider two scale principles respecting different scenarios: **(1) Uniform**. Workers are with very similar performance; The time a worker needs to complete a local iteration of BP+FP, follows the uniform distribution of $U[0.9,1.1] \times 1.5$ s, with an average value of 1.5 s. This is the default setting in tests. **(2) With stragglers**. Workers might suffer from a remarkably larger training time, with a low probability. For this type, we consider that the computation time of a local iteration follows the distribution tested on a standard Google Cloud GPU instance using the dataset of ImageNet, reported by [32]. To be comparable with the case of uniform, the expected value of the generated computation times keeps 1.5 s, as Fig. 8 shows.

5.2. Results

5.2.1. React to the change of bandwidth

To check the ability of AQGB on reacting to the change of bandwidth, we proactively reconfigure the available link bandwidth that it would use and record the compression ratio AQGB estimated via

[3] Concurrent transfers share link capacities with max–min fairness.

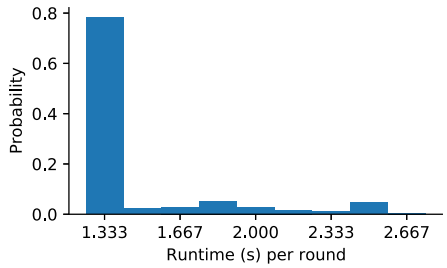


Fig. 8. A synthesized runtime distribution of training workers based on that reported by [32].

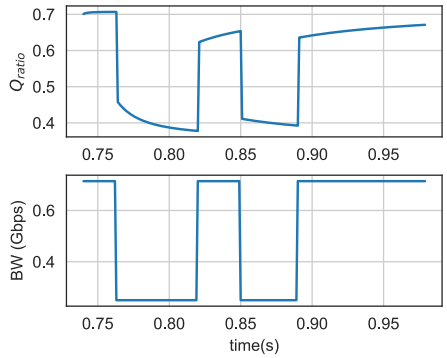
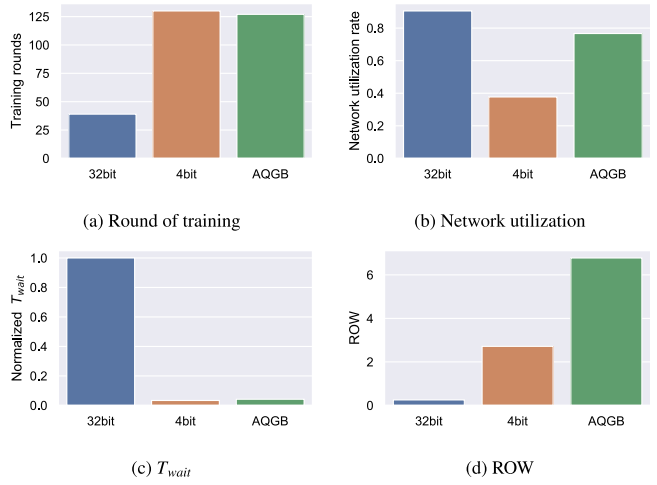


Fig. 9. An example shows that AQGB could adjust the level of quantization respecting the change of available bandwidth.



(a) Round of training (b) Network utilization

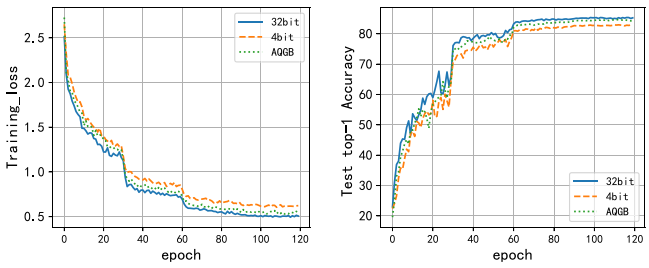
(c) T_{wait} (d) ROW

Fig. 10. With adaptive quantization, AQGB reduces T_{wait} while making efficient usage of bandwidth. Distinguished from the metric of T_{wait}, which only reflects the time that training workers are blocked by communication, ROW could capture the overlap between computation and communication, thus preventing AQGB from making over-killed quantization.

$\max(^{b(t)(t^{BP}_{n+1-i(t)}-t)}/_{u(t)}, 0.375)$. Fig. 9 shows an instance we observed when 8 workers are training ResNet50. Despite that the absolute values are related to the instance, we do consistently observe that AQGB is able to adjust the quantization ratio (i.e., Q_{ratio} defined by Eq. (3) and shown in Fig. 9) respecting the network status.

5.2.2. Acceleration of the training iteration

Now, we check the advantage of the proposed ROW and AQGB. Fig. 10(a) shows that compared to the scheme of no-quantization (i.e., 32 bit), AQGB enables workers to obtain more training rounds in the same specified time. Consistent with the results of Fig. 12, when



(a) Speed of convergence (b) Top-1 accuracy

Fig. 11. Training convergence speed and model accuracy for DDL tasks that use 8 simulated workers to train the model of ResNet50 over the CIFAR-10 dataset, with no-quantization, fixed-quantization, and AQGB, where the EF is not enabled.

Table 2
ResNet50 upon CIFAR-10.

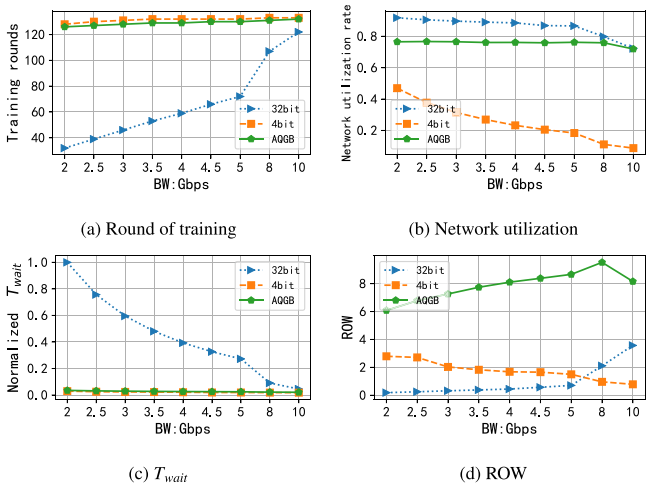
Quantization scheme	Top-1 accuracy	Convergence rate	Acceleration (CCO)	Actual speedup
None (i.e., FP32)	85.36%	1(Baseline)	1(Baseline)	1(Baseline)
4 bit QSGD	82.94%	0.76×	3.33×	2.54×
AQGB	84.69%	0.97×	3.26×	**3.15**×

AQGB limits the quantization level to be no less than 4 bit, workers could iterate the training very fast, close to the case when fixed 4bit-QSGD quantization is employed. For example, in the case shown in Fig. 10, without quantization, workers would iterate 39 rounds within 200 s; while with AQGB, workers iterate 127 rounds, close to the 130 rounds workers achieve under 4 bit quantization. Despite iterating the most rounds, as Fig. 10(b) shows, 4bit-QSGD quantization achieves the worst network bandwidth utilization, less than half of that achieved by AQGB. In contrast, both AQGB and the no-quantization scheme achieve much higher utilizations of about 77% and 90%, with a gap of about 13%. Higher network utilization means fewer data are dropped, yielding better quantization quality thus bringing possible benefits to the model accuracy or training convergence [16].

Consistent with the results in Fig. 10(a), as Fig. 10(c) shows, the T_{wait} achieved by AQGB is close to that of 4bit-based quantization, significantly outperforming the case when no quantization is employed. Besides, we also plot their ROW values as Fig. 10(d) shows. Obviously, compared to T_{wait}, ROW is a better metric: besides reflecting the time that training workers are blocked by gradient communication (as T_{wait} does), it also captures the overlap between computation and communication. Thus, using ROW as the optimization goal, quantization schemes like AQGB would control the level of compression properly without over-killing.

5.2.3. Optimized convergence speed and final accuracy

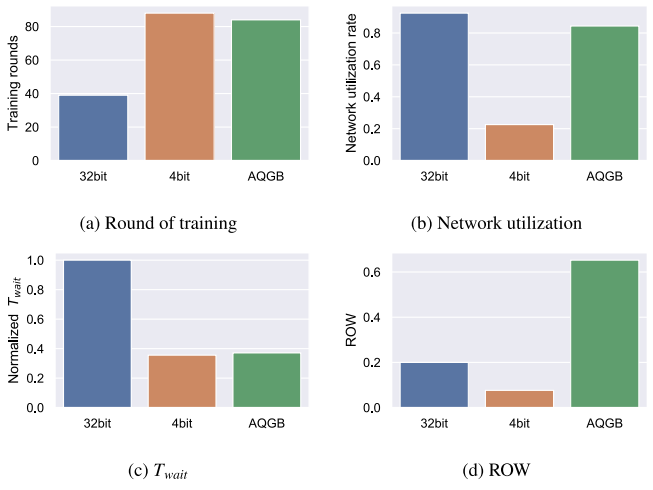
Fig. 11 illustrates the model performance in a simulated DDL scenario. As Fig. 11(a) shows, with AQGB, the convergence rate of distributed training is slightly lower than the case of no-quantization but much higher than the case of 4 bit quantization (in terms of the number of training rounds to converge), while Fig. 11(b) shows that the accuracy achieved by AQGB-enabled training is almost comparable to the case of no quantization; and as shown in Table 2, the gap between the two cases is only 0.67%. By combining the observed improvements in terms of convergence rate and CCO optimized by AQGB, we further estimate the actual speedup ratio AQGB could achieve on distributed training: compared to no-quantization, AQGB could reduce the training time by about 3.15×, while keeping the decrease of the top-1 accuracy about 0.67%. Even compared to 4bit-QSGD quantization, AQGB achieves an improvement of 3.15×, which is significantly higher than the 2.54× achieved by 4 bit fixed quantization, yielding about 24% improvement in the acceleration ratio and a 2.75% increase in model



(a) Round of training (b) Network utilization

(c) T_{wait} (d) ROW

Fig. 12. Impacts of link bandwidth.



(a) Round of training (b) Network utilization

(c) T_{wait} (d) ROW

Fig. 13. Performances when the time cost of local training follows the distribution of Fig. 8.

accuracy. It can be seen that AQGB is all-around superior compared to traditional fixed quantization schemes. Taking advantage of the awareness of the dynamics of network bandwidth, AQGB achieves higher-quality gradient transmission with almost no reduction in the number of iterations per time. In summary, given that AQGB has a faster convergence rate as well as a smaller risk of accuracy loss, when developing new models, at the early stages of the training where fast convergence is desired, it is promising to use AQGB instead of a fixed gradient quantization (e.g., QSGD).

5.2.4. Impacts of bandwidth

Next, we study the influence of available bandwidth by increasing the link capacities from 2 Gbps to 10 Gbps and rerun the tests. As Fig. 12(b) shows, with the increase in bandwidth, the observed bandwidth utilization decrease for all schemes, including the scheme without quantization. As expected, such a result implies that the bottleneck effects of communication are alleviated; therefore, due to the reduction of T_{wait} as Fig. 12(c) shows, workers could conduct more training rounds as confirmed by Fig. 12(a). Results also imply that, for both AQGB and the 4 bit quantization scheme, the improvements of the training round are trivial. This is because, with quantization, they already relieve the bottleneck efforts of gradient communication. However, as Figs. 12(b) and 12(d) show, unlike using the fixed 4 bit quantization, the adaptive design of AQGB enables it to make effective use of the available bandwidth, leading to higher network utilization and ROW. Moreover, from Fig. 12(d), we also observe that, for AQGB, once communication is not the bottleneck, the value of ROW also decreases with growing bandwidth. This is reasonable since the overlap between computation and communication (i.e., the numerator in the definition of ROW) is decreased.

5.2.5. Impacts of stragglers

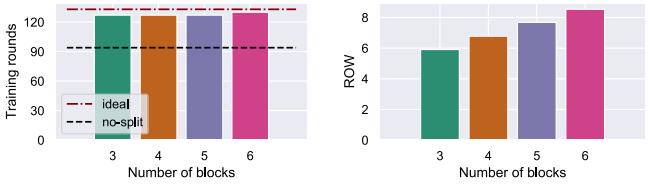
To further investigate the influence of the straggler in the training, we assume that the time cost a worker needs to complete a round of local training following the distribution shown in Fig. 8, belonging to the range of [1.33, 2.67] s, with the expected value of 1.5 s. In case a worker spends noticeably more than 1.5 s (e.g., 2.0 s) to complete a round of training, we treat it as a straggler in this round. The appearance of such stragglers enforces other normal workers to wait for their completion for the transmission of gradients in each round of synchronization. As Fig. 13(a) shows, due to the presence of straggler, the improved number of training rounds for both 4 bit quantization and AQGB are only about 125.6% and 118.0%, respectively, when compared to no-quantization. Such achieved improvements in training rounds are much less than those obtained in the uniform case without stragglers (i.e., Fig. 10(a)). As for the network utilization (Fig. 13(b)), the results are similar to the uniform scenario. And because of stragglers, the room for the optimizations of both T_{wait} and ROW is limited, Compared to benchmarks, performance drops by almost an order of magnitude (Figs. 13(c) and 13(d)). Nevertheless, compared to no-quantization and 4bit-quantization, AQGB could make more efficient use of the underlying network by maximizing the CCO.

5.2.6. Impacts of block splitting

To investigate the impacts of the block-splitting scheme, we control the number of split blocks from 3 to 6 and rerun the tests. As we have mentioned, in the case of 4 blocks, the model is divided following the scheme specified by Table 1. Based on this, we merge the 3rd and 4th blocks to generate the case of 3-block-splitting, and split the 4th block (e.g., conv5_2-fc1000, Table 1) to the two parts of conv5_2, and conv5_3-fc1000, to generate the case of 5-block-splitting. As for the case of 6-block-splitting, it is generated by continuing to regroup the first 2 blocks (i.e., conv1-conv4_6) into 3 new blocks, conv1-conv3_4, conv4_1-conv4_3, and conv4_4-conv4_6. Fig. 14 shows both the training rounds and average ROW values AQGB achieved in a period of 200 s. Here, *no-split* refers to the (lower) case where the entire model is not split; and *ideal* refers to the (upper) ideal-yet-unachievable situation in which perfect CCO is obtained (i.e., $T_{wait} = 0$, all communication is masked). Thus, the gap between *no-split* and *ideal* shows the room for the optimization of the iteration speed of the training. As indicated by the results, despite splitting the model into more blocks could increase the training iterations a little, it distinctly improves the CCO, as the value of ROW continues to grow. Thus, for communication optimization, workers prefer a fine-grained split of the model. However, in practice, how a model could be split also highly depends on its structure, and small blocks generally introduce additional communication overhead. Thus, future studies are needed to explore generic principles and guidelines for the optimization of block splitting.

6. Related work

The key idea of our proposed scheme is to maximize the CCO for DDL in dynamic environments with adaptive lossy gradient quantization. For both overlapping the communication with computation and lossy gradient compression, there are a great many various proposals [5,6]. In the following, we briefly discuss some recent advances related to AQGB.



(a) Rounds of training (b) ROW

Fig. 14. Impacts of the number of split blocks on the performance of AQGB. Results imply that, despite splitting the model into more blocks only increasing the training iterations a little, it distinctly improves the CCO, as the value of ROW continues to grow.

6.1. Overlapping communication with computation

Different from ooo [28] that reorders the executions of back-prop progress, iPart [11] overlaps the gradient communication with backward computation and parameter communication with forward computation for parameter server based DDL, by partitioning the involved communication and computation in suitable partition sizes. AccTFM [42] employs similar but more sophisticated designs for the distributed training of specialized Transformer-based DNNs, in which, both sparsification and quantization methods are also embodied to reduce the involved traffic volume. Also aiming at the goal of optimizing the CCO, Prophet [31] directly use the wait time as the metric and tries to minimize it by reordering the delivery of gradients in block-wise manners [31]. Distinguished from them, in this paper, we propose the metric of ROW to explicitly measure the CCO for DDL, which is a better metric for the optimization objective as we have discussed in Section 2.

6.2. Lossy gradient quantization

As a lossy compression technique, quantizing float32 gradients into fewer bits could reduce the traffic volume, with the possible cost of slowed convergence speed or reduced model accuracy. For example, researchers have shown recently that, in theory, worse compression quality leads to slower convergence [16]. However, in practice, the convergence of a model on a given dataset is determined by a lot of hyper-parameters jointly, such as the batch size, training algorithm, and learning rate [34]. Thus, there does not exist a clear and fixed relationship between a DDL task's convergence speed and the compression quality, or more specifically, the quantization level, it has taken advantage of [43]. Besides determining the level of quantization in advance [5], some recent proposals have proposed adaptive designs that could adjust the quantization settings respecting the gradients updates and other dynamic factors during training [26,27]. However, none of them have taken the dynamic available bandwidth into account and AQGB overcomes this type of drawback. Besides quantization, sparsification (e.g., Top-k, random-k) is another type of lossy compression scheme widely employed to reduce the traffic load of gradient synchronization for DDL [5,19]. Some recent proposals also show that these two types of designs can work together for communication optimization [44]. Currently, AQGB only supports adaptive quantization; extending it to support adaptive joint-sparsification-and-quantization communication optimization yields attractive future directions.

7. Conclusion and future work

In this paper, we revisit the idea of using gradient quantization techniques to tame the communication bottlenecks involved in DDL. Different from prior solutions, our proposed solution, AQGB, embodies two novel designs. Firstly, instead of employing fixed quantization designs, it dynamically adapts the level of quantization respecting the training and transmission progress, bringing benefits to the convergence and quality of the training [5,15–17]. And secondly, rather than just reducing the time training workers take for the completion of a round of model synchronization, it purses the explicit goal of maximizing their ROW (the Ratio of Overlap time to Wait time), fully releasing the power of overlapping communication with computation for DDL [4]. Detailed trace-based performance studies confirm that AQGB can optimize the utilization of both the computation and network resources, thus improving the DDL system performance significantly.

Regarding the practical deployment, AQGB can be implemented as a part of the communication library [45], and using the existing reliable transport mechanisms like MPI_Bcast and reliable multicast transport protocol(s) [46] for quantized gradient broadcast. Although the computations involved are not particularly complicated, a full-fledged implementation of AQGB still requires substantial future engineering effort. Besides, as a generic approximate gradient synchronization framework that could adjust the level of quantization concerning the training progress and network state, AQGB supports any other quantization ratio control and multi-level quantization schemes besides the ones specified in Sections 4.3 and 4.4. Accordingly, it is possible to make joint usage of AQGB and other quantization schemes like QSGD [18], DAdaQuant [47], DQ-SGD [48], and others [26,27]. However, non-trivial modifications on both AQGB and these schemes are needed to achieve the goal, as they were not originally designed for that purpose, having not supported adaptive multi-level quantization yet. We leave these for future work.

CRediT authorship contribution statement

Shouxi Luo: Writing – review & editing, Writing – original draft, Supervision, Resources, Project administration, Methodology, Funding acquisition, Conceptualization. **Xue Liu:** Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Data curation. **Ke Li:** Writing – review & editing. **Huanlai Xing:** Writing – review & editing. **Xu Zhang:** Writing – review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

References

[1] X. Liu, S. Luo, K. Li, H. Xing, Approximate gradient synchronization with AQGB, in: Proceedings of the 6th APNet, ACM, 2022, pp. 101–102.
[2] M. Li, D.G. Andersen, J.W. Park, A.J. Smola, A. Ahmed, V. Josifovski, J. Long, E.J. Shekita, B.-Y. Su, Scaling distributed machine learning with the parameter server, in: Proceedings of the 11th OSDI, USENIX Association, USA, 2014, pp. 583–598.
[3] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E.P. Xing, J.E. Gonzalez, I. Stoica, Alpa: Automating inter- and intra-operator parallelism for distributed deep learning, in: Proceedings of the 16th OSDI, 2022, pp. 559–578.
[4] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, C. Guo, A generic communication scheduler for distributed DNN training acceleration, in: Proceedings of the 27th SOSP, ACM, 2019, pp. 16–29.
[5] H. Xu, C.-Y. Ho, A.M. Abdelmoniem, A. Dutta, E.H. Bergou, K. Karatsenidis, M. Canini, P. Kalnis, GRACE: A compressed communication framework for distributed machine learning, in: Proceedings of the 41st ICDCS, 2021, pp. 561–572.
[6] S. Shi, Z. Tang, X. Chu, C. Liu, W. Wang, B. Li, A quantitative survey of communication optimizations in distributed deep learning, IEEE Netw. 35 (3) (2021) 230–237.

[7] S.H. Hashemi, S. Abdu Jyothi, R. Campbell, Tictac: Accelerating distributed deep learning with communication scheduling, in: Proceedings of the 2nd SysML Conference, Vol. 1, 2019, pp. 418–430.

[8] M. Li, R.B. Basat, S. Vargaftik, C. Lao, K. Xu, M. Mitzenmacher, M. Yu, THC: Accelerating distributed deep learning using tensor homomorphic compression, in: Proceedings of the 21st NSDI, USENIX Association, Santa Clara, CA, 2024, pp. 1191–1211.

[9] S. Luo, X. Yu, K. Li, H. Xing, Releasing the power of in-network aggregation with aggregator-aware routing optimization, IEEE/ACM Trans. Netw. 32 (5) (2024) 4488–4502.

[10] J. WANG, Y. Lu, B. Yuan, B. Chen, P. Liang, C.D. Sa, C. Re, C. Zhang, Cocktailsgd: Fine-tuning foundation models over 500Mbps networks, in: Proceedings of the 40th ICML, PMLR, 2023.

[11] S. Wang, A. Pi, X. Zhou, J. Wang, C.-Z. Xu, Overlapping communication with computation in parameter server for scalable DL training, IEEE Trans. Parallel Distrib. Syst. 32 (9) (2021) 2144–2159.

[12] J. Wu, W. Huang, J. Huang, T. Zhang, Error compensated quantized SGD and its applications to large-scale distributed optimization, in: Proceedings of the 35th ICML, Vol. 80, PMLR, 2018, pp. 5325–5333.

[13] C. Yu, H. Tang, C. Renggli, S. Kassing, A. Singla, D. Alistarh, C. Zhang, J. Liu, Distributed learning over unreliable networks, in: Proceedings of the 36th ICML, Vol. 97, PMLR, 2019, pp. 7202–7212.

[14] P. Zhou, X. He, S. Luo, H. Yu, G. Sun, JPAS: Job-progress-aware flow scheduling for deep learning clusters, J. Netw. Comput. Appl. 158 (2020) 102590.

[15] J. Xia, G. Zeng, J. Zhang, W. Wang, W. Bai, J. Jiang, K. Chen, Rethinking transport layer design for distributed machine learning, in: Proceedings of the 3rd APNet, ACM, 2019, pp. 22–28.

[16] A. Koloskova*, T. Lin*, S.U. Stich, M. Jaggi, Decentralized deep learning with arbitrary communication compression, in: Proceedings of ICLR, 2020.

[17] S.U. Stich, On communication compression for distributed optimization on heterogeneous data, 2020, CoRR abs/2009.02388.

[18] D. Alistarh, D. Grubic, J. Li, R. Tomioka, M. Vojnovic, QSGD: Communication-efficient SGD via randomized quantization and encoding, in: Proceedings of the 31st Annual Conference on Neural Information Processing Systems, NIPS, Curran, 2017, pp. 1710–1721.

[19] A.M. Abdelmoniem, M. Canini, DC2: Delay-aware compression control for distributed machine learning, in: Proceedings of INFOCOM, 2021, pp. 1–10.

[20] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, J.S. Rellermeyer, A survey on distributed machine learning, ACM Comput. Surv. 53 (2) (2020).

[21] A. Mathkar, V.S. Borkar, Distributed reinforcement learning via gossip, IEEE Trans. Autom. Control 62 (3) (2017) 1465–1470.

[22] S. Luo, P. Fan, K. Li, H. Xing, L. Luo, H. Yu, Efficient parameter synchronization for peer-to-peer distributed learning with selective multicast, IEEE Trans. Serv. Comput. 18 (1) (2025) 156–168.

[23] S. Luo, R. Wang, H. Xing, Efficient inter-datacenter AllReduce with multiple trees, IEEE Trans. Netw. Sci. Eng. 11 (5) (2024) 4793–4806.

[24] J. Xin, M. Canini, P. Richtárik, S. Horváth, Global-QSGD: Practical floatless quantization for distributed learning with theoretical guarantees, 2023, arXiv:2305.18627 [cs, stat].

[25] Challenges of quantization in machine learning (ML), 2023, https://iq.opengenus.org/challenges-of-quantization//. (Online Accessed 31 July 2023).

[26] Y. Mao, Z. Zhao, G. Yan, Y. Liu, T. Lan, L. Song, W. Ding, Communication-efficient federated learning with adaptive quantization, ACM Trans. Intell. Syst. Technol. 13 (4) (2022).

[27] G. Yan, T. Li, S.-L. Huang, T. Lan, L. Song, AC-SGD: Adaptively compressed SGD for communication-efficient distributed learning, IEEE J. Sel. Areas Commun. 40 (9) (2022) 2678–2693.

[28] H. Oh, J. Lee, H. Kim, J. Seo, Out-of-order backprop: An effective scheduling technique for deep learning, in: Proceedings of the 17th EuroSys, ACM, 2022, pp. 435–452.

[29] S. Wang, D. Li, J. Geng, Geryon: Accelerating distributed CNN training by network-level flow scheduling, in: Proceedings of INFOCOM, 2020, pp. 1678–1687.

[30] T. Liu, T. Miao, Q. Wu, Z. Li, G. He, J. Wu, S. Zhang, X. Yang, G. Tyson, G. Xie, Modeling and optimizing the scaling performance in distributed deep learning training, in: Proceedings of the ACM Web Conference, WWW, ACM, 2022, pp. 1764–1773.

[31] Z. Zhang, Q. Qi, R. Shang, L. Chen, F. Xu, Prophet: Speeding up distributed DNN training with predictable communication scheduling, in: Proceedings of the 50th ICPP, ACM, New York, NY, USA, 2021.

[32] S. Li, T. Ben-Nun, S.D. Girolamo, D. Alistarh, T. Hoefler, Taming unbalanced training workloads in deep learning with partial collective operations, in: Proceedings of the 25th PPoPP, ACM, New York, NY, USA, 2020, pp. 45–61.

[33] A. Eisenman, K.K. Matam, S. Ingram, D. Mudigere, R. Krishnamoorthi, K. Nair, M. Smelyanskiy, M. Annavaram, Check-n-run: a checkpointing system for training deep learning recommendation models, in: Proceedings of the 19th NSDI, Renton, WA, 2022, pp. 929–943.

[34] L. Mai, G. Li, M. Wagenländer, K. Fertakis, A.-O. Brabete, P. Pietzuch, Kungfu: Making training in distributed machine learning adaptive, in: Proceedings of the 14th OSDI, USENIX Association, 2020, pp. 937–954.

[35] S. Luo, R. Wang, K. Li, H. Xing, Efficient cross-cloud partial reduce with CREW, IEEE Trans. Parallel Distrib. Syst. 35 (11) (2024) 2224–2238.

[36] IEEE standard for floating-point arithmetic, in: IEEE Std 754-2019 (Revision of IEEE 754-2008), 2019, pp. 1–84.

[37] X. Sun, N. Wang, C.-y. Chen, J.-m. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. Srinivasan, K. Gopalakrishnan, Ultra-low precision 4-bit training of deep neural networks, in: Proceedings of the 34th NIPS, 2020.

[38] S. Luo, P. Fan, H. Xing, H. Yu, Meeting coflow deadlines in data center networks with policy-based selective completion, IEEE/ACM Trans. Netw. 31 (1) (2023) 178–191.

[39] M. Khani, M. Ghobadi, M. Alizadeh, Z. Zhu, M. Glick, K. Bergman, A. Vahdat, B. Klenk, E. Ebrahimi, Sip-ML: high-bandwidth optical network interconnects for machine learning training, in: Proceedings of the ACM SIGCOMM Conference, 2021, pp. 657–675.

[40] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, T. Krishna, SCALE-sim: Systolic CNN accelerator simulator, 2019, arXiv:1811.02883.

[41] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of CVPR, 2016, pp. 770–778.

[42] Z. Zeng, C. Liu, Z. Tang, K. Li, K. Li, Acctfm: An effective intra-layer model parallelization strategy for training large-scale transformer-based models, IEEE Trans. Parallel Distrib. Syst. 33 (12) (2022) 4326–4338.

[43] A. Dutta, E.H. Bergou, A.M. Abdelmoniem, C.-Y. Ho, A.N. Sahu, M. Canini, P. Kalnis, On the discrepancy between the theoretical analysis and practical implementations of compressed communication for distributed deep learning, Proc. AAAI 34 (04) (2020) 3817–3824.

[44] D. Basu, D. Data, C. Karakus, S.N. Diggavi, Qsparse-local-SGD: Distributed SGD with quantization, sparsification, and local computations, IEEE J. Sel. Areas Inf. Theory 1 (1) (2020) 217–226.

[45] A. Weingram, Y. Li, H. Qi, D. Ng, L. Dai, X. Lu, xCCL: A survey of industry-led collective communication libraries for deep learning, J. Comput. Sci. Tech. 38 (1) (2023) 166–195.

[46] S. Luo, H. Yu, K. Li, H. Xing, Efficient file dissemination in data center networks with priority-based adaptive multicast, IEEE J. Sel. Areas Commun. 38 (6) (2020) 1161–1175.

[47] R. Hönig, Y. Zhao, R. Mullins, DAdaquant: Doubly-adaptive quantization for communication-efficient federated learning, in: Proceedings of the 39th ICML, Vol. 162, 2022, pp. 8852–8866.

[48] G. Yan, S.-L. Huang, T. Lan, L. Song, DQ-SGD: Dynamic quantization in SGD for communication-efficient distributed learning, in: Proceedings of the 18th MASS, IEEE, 2021, pp. 136–144.