SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

June 2023, Vol. 66 162102:1–162102:17
https://doi.org/10.1007/s11432-021-3532-8

Accelerate distributed deep learning with cluster-aware sketch quantization

Keshi GE, Yiming ZHANG, Yongquan FU, Zhiquan LAI, Xiaoge DENG & Dongsheng LI*

College of Computer, National University of Defense Technology, Changsha 410073*, China*

Received 5 September 2021/Revised 9 December 2021/Accepted 10 May 2022/Published online 22 May 2023

Abstract Gradient quantization has been widely used in distributed training of deep neural network (DNN) models to reduce communication cost. However, existing quantization methods overlook that gradients have a nonuniform distribution changing over time, which can lead to significant compression error that not only increases the number of training iterations but also requires a higher number of quantization bits (and consequently higher delay for each iteration) to keep the validation accuracy as high as the original stochastic gradient descent (SGD) approach. To address this problem, in this paper we propose cluster-aware sketch quantization (CASQ), a novel sketch-based gradient quantization method for SGD with convergence guarantees. CASQ models the nonuniform distribution of gradients via clustering, and adaptively allocates appropriate numbers of hash buckets based on the statistics of different clusters to compress gradients. Extensive evaluation shows that compared to existing quantization methods, CASQ-based SGD (i) achieves the same validation accuracy when decreasing quantization level from 3 bits to 2 bits, and (ii) reduces the training time to convergence by up to 43% for the same training loss.

Keywords distributed training, deep learning, communication, sketch, quantization

Citation Ge K S, Zhang Y M, Fu Y Q, et al. Accelerate distributed deep learning with cluster-aware sketch quantization. Sci China Inf Sci, 2023, 66(6): 162102, https://doi.org/10.1007/s11432-021-3532-8

1 Introduction

Stochastic gradient descent (SGD) [1–3] has been widely used for distributed training deep neural network (DNN) models [4]. In data-parallel distributed SGD, however, iterative gradient synchronization is conducted between worker nodes and parameter servers [5] or among worker nodes, which leads to heavy network transmission, especially for training complex models [6–11].

Researchers have proposed many gradient compression methods to reduce communication costs. Quantization is one of the most effective ones. Quantization compresses the stochastic gradients by representing them with a few bits. Existing quantization methods map the normalized gradients to a set of fixed quantization levels prior to training, such as QSGD [12] and TernGrad [13]. However, these methods overlook a vital property of gradients in deep learning models that gradients conform to a nonuniform distribution (as shown in Figure 1) which changes throughout the training process [14]. The nonuniform distribution of gradients tends to cause significant compression errors. As illustrated in Figure 1(a), the large gradients, whose absolute value is far away from 0, are rare and widely spread. This phenomenon means the quantization error on large gradients tends to be more significant, due to the large interval. As demonstrated in Figure 2(a), existing methods treat large and small gradients equally and quantize them with the same number of bits, which leads to a significant gradient compression error. The high compression error may delay the convergence of SGD and even decrease the accuracy of models [14–16]. Consequently, these methods have to train the model with more steps and use more quantization levels, which decreases the compression rate.

* Corresponding author (email: dsli@nudt.edu.cn)

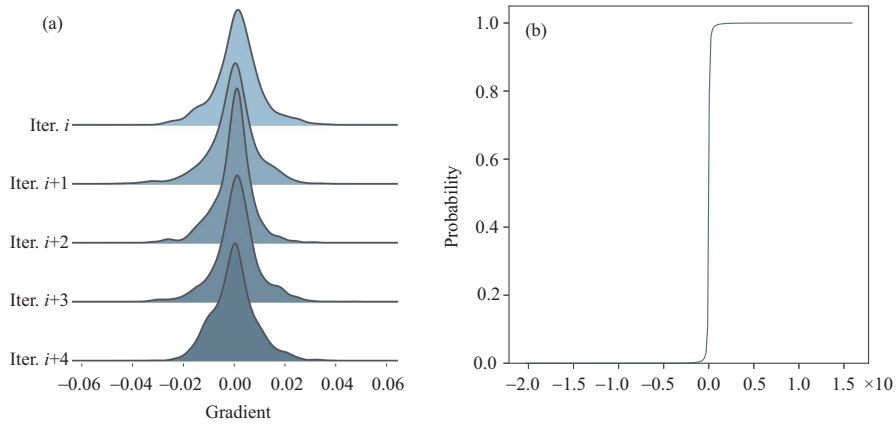

Figure 1 (Color online) Most of the gradients are very close to zero, and the large gradients are rare. (a) The non-uniform distribution curve of a convolutional layer over consecutive five training iterations; (b) the cumulative distribution functions curve of ResNet34 model's gradient.

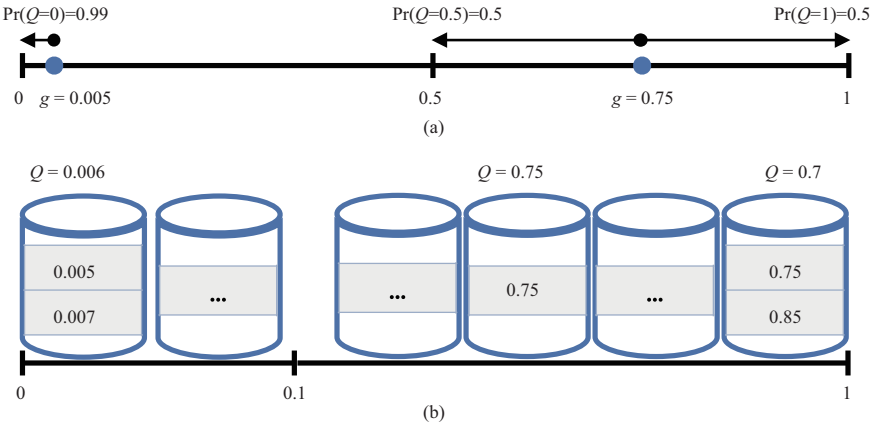

Figure 2 (Color online) (a) Gradient quantization with three uniform levels 0,0.5,1. The gradient value 0.75 is quantized to 0.5 or 1 with a probability of 0.5, the error is 0.25. Gradient 0.005 is quantized to 0 with a probability of 0.99. Thus, large gradients usually have large compression errors. $\Pr(\cdot)$ is the probability of rounding the gradient g to the quantized value Q. (b) CASQ quantizes a gradient as the average value Q in its bucket. The gradients are partitioned according to their value and randomly mapped to a bucket. Some large gradients can be losslessly compressed, such as 0.75.

To address this problem, we introduce sketch to gradient quantization to minimize the compression error. Sketch is a kind of algorithm widely used to estimate the distribution of a data stream. We propose cluster-aware sketch quantization (CASQ), a novel sketch-based gradient quantization method that builds a set of hash bucket arrays (sketch) to approximate the nonuniform distribution of the gradient.

On each worker node, CASQ first clusters the gradients into groups. Each group has a designated hash bucket array, and gradients in this group will be randomly hashed to the buckets. The cluster-aware sketch, consisting of the cluster results and the buckets, is gathered by the parameter server. Then, the average of values within the mapped bucket is returned as the decompressed gradient on the server nodes. As illustrated in Figure 2(b), the estimates are closer to the gradients compared with existing quantization methods.

The large gradients are essential yet rare in the nonuniform distribution, as shown in Figure 1(b). CASQ leverages this phenomenon by adaptively assigning more buckets for larger gradients groups. This technique ensures more accurate estimations for (possibly) smaller-sized but more essential groups. Therefore, CASQ can reduce the compression error with a small-sized sketch. With the reduced error, CASQ can obtain the same model quality with fewer iterations and fewer bits to quantization, consequently reducing the training time.

We have realized CASQ and conducted extensive experiments on residual networks on CIFAR-10/100 and ImageNet in a cluster with up to 24 GPUs. The results demonstrate that (compared with existing quantization methods) CASQ (i) most significantly reduces the compression error compared with the existing quantization methods, (ii) achieves comparable model quality against the full-precision training,

and improves the validation accuracy by up to 4.9% using 2 bits, (iii) achieves up to 5.4× speedup compared with distributed SGD, and (iv) decreases up to 43% of the training time for the same training loss.

We summarize the contributions of this paper as follows.

• We identify the gradient compression error problem of existing quantization methods caused by nonuniform and changing gradient distribution and design a novel cluster-aware-sketch based gradient compression scheme to reduce the compression error.

• We theoretically analyze the gradient compression error of CASQ, proving the convergence rate of CASQ to match that of vanilla distributed SGD.

• We implement CASQ[1)] in the large-scale GPU cluster, and conduct extensive experiments to demonstrate its advantage (speed and accuracy) over existing quantization methods.

2 Preliminaries

In this section, we first show how the data parallel distributed SGD works in the parameter server architecture [17]. We also describe the gradient quantization procedures and introduce a traditional sketch algorithm based on stochastic projection.

2.1 Data parallel distributed SGD

In distributed setting, worker node $i \in \{1, \ldots, W\}$ samples a mini-batch $X_t^i \subset \{x_1, \ldots, x_n\}$, computes local gradient $g_t^i = \frac{1}{|X_t^i|} \sum_{x_j \in X_t^i} \nabla f_i(\boldsymbol{\omega}_t, x_j)$ with back-propagation. Then they synchronize these gradients, typically by aggregating and broadcasting through the server node. Finally each worker node updates parameters $\boldsymbol{\omega}_{t+1}$ as

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t - \eta_t \nabla f_i(\boldsymbol{\omega}_t) \tag{1}$$

to trigger the next iteration, where η_t denotes the learning rate in iteration t, $f_i(\cdot)$ denotes the loss function of node i.

2.2 Gradient quantization

The gradient synchronization requires the workers to suspend computing before next iteration; this lead to significant communication cost. Quantization is a mainstream technique to compress the gradient. Let $\boldsymbol{\ell} = \{\ell_0, \ell_1, \ldots, \ell_s, \ell_{s+1}\}$ denote the quantization levels, which are sorted in an ascending order, and ℓ_0, ℓ_{s+1} are the minimum and maximum in the gradient vector. Each gradient coordinate g_i is randomly mapped to a quantization level ℓ_k or $\ell_{k+1} \in \boldsymbol{\ell}$ such that $\ell_k \leqslant g_i \leqslant \ell_{k+1}$; then g_i is represented as k or $k+1$, and encoded with $\lceil \log(s+2) \rceil$ bits. Although quantization methods demonstrate the communication reduction results, several researchers observe that it deteriorates the convergence [12, 14, 16]. In these studies, the convergence rates delay as the compression rate increases. The model accuracy may degrade, even divergence, when quantizing gradients with extremely fewer bits, such as 2-bits quantization.

2.3 Sketch algorithm

Sketch is one of the most popular communication traffic statistical algorithms in network-flow monitoring [18, 19]. Here we take the gradient estimation as an example to depict the typical sketch process. Assume that a sketch consists of one hash bucket array for ease of analysis. The algorithm randomly maps the ith coordinate of the gradient vector to the jth hash bucket and returns the sum of all gradients within the jth bucket as the estimated gradient for i. However, this method incurs significant errors and is not fit to compress the gradient. Previous work [20] proves that the gradients variance is proportional to the number of elements in each bucket, which is significantly high. The proposed method, cluster-aware sketch quantization, designs a novel sketch and combines it with quantization to reduce the variance.

1) Open source code. https://github.com/GeKeShi/CASQ.

3 Cluster-aware sketch quantization

In Subsection 3.1, we first discuss the motivations of CASQ, i.e., decreasing the compression error with the cluster-aware sketch. Then we illustrate the compression and decompression procedure of our method (Subsection 3.2). We also show that the K-means algorithm can find the optimal cluster assignments for CASQ (Subsection 3.3). Finally, We present the implementation of CASQ based SGD algorithm by taking the typical parameter server as an example (Subsection 3.4).

3.1 Why use cluster-aware sketch

We collect the gradients when training deep learning models and show the distribution in Figure 2. It is clear that gradients are non-uniformly distributed; e.g., most of the gradients are very close to zero and pile up. In contrast, the large gradients are rare and randomly spread between the quantization levels. Although large gradients are rare, they must be more carefully treated in compression methods than the small ones. For example, when there are three quantization levels, $\{0, 0.5, 1\}$, gradient 0.005 is quantized to 0, the compression error is 0.005, gradient 0.75 is quantized to 1, whose compression error is 50× larger than the former one. Thus, large gradients usually lead to significant compression errors. However, current quantization methods ignore this fact. They treat the small and large gradients equally. Consequently, their compression error is significant, and the convergence may be delayed. The model accuracy may even be downgraded when these methods' compression rate is extremely high.

The highly skewed gradient distribution motivates us to partition gradients into clusters according to their values and leverage the gradient clusters' information to reduce the compression error. Specifically, we propose a cluster-aware-sketch quantization, which classifies the gradients with the clustering method and compresses different gradient clusters with a different number of hash buckets. CASQ uses K-means as the clustering algorithm to find the optimal clustering assignments, which will be explained later. We use the hash buckets to store the gradient values. Because the large gradients are rare, these gradients have the opportunity to be losslessly stored. Thus, CASQ can reduce the compression error to a satisfying level with a small number of hash buckets.

3.2 Description of cluster-aware sketch quantization

CASQ contains compression and decompression procedures. Gradients are compressed into cluster-aware sketch, which consists of the gradient clustering assignment and the hash buckets. The hash buckets are adaptively allocated for different gradient clusters.

Compression. The compression procedures of CASQ are illustrated in the left part of Figure 3. Let $g, \hat{g} \in \mathbb{R}^N$ denote the vectors of gradients and the compressed gradients, respectively. Let m, N denote the number of buckets and gradients, respectively. It contains the steps 1–3 in Figure 3.

(1) It clusters each gradient to one of the $K = 2^b$ (the number of quantization levels) clusters in the first step. The clustering step derives an assignment matrix $C \in \{0,1\}^{N\times K}$, where $C(i,k) = 1$ $(i \leqslant N, k \leqslant K)$ if the ith gradient $g(i)$ is assigned to kth cluster and $C(i,k) = 0$ otherwise. Note that the kth cluster is not only the nearest in the distance for the $g(i)$ but also has the same sign as this gradient.

(2) Gradient $g(i)$ is randomly mapped to the bucket $h(i)$ in the kth bucket array by the hash function h.

(3) Adding $g(i)$ upon the current value of the bucket in the insertion phase. Let $A \in \{0,1\}^{N\times m}$ denote the mapping matrix where the entry $A(i, h(i))$ is 1, and other entries are 0. This process can be mathematically represented as a stochastic projection model $A^{\mathrm{T}}g$.

Decompression. The decompression is shown in the right part of Figure 3. It contains the last two step.

(4) Finding the bucket array corresponding to the jth gradient based on the cluster assignment $C(j)$.

(5) Querying $g(j)$ by averaging the sum of values in the bucket $h(j)$. h is the hash function used in the compression and decompression process.

In the stochastic projection model mentioned above, the querying phase can be formulated as $\hat{g} = AA^{\mathrm{T}}g$. Let $g_k = \mathrm{diag}(C(:,k))g$ denote the gradients in cluster k. Let $A_k = \mathrm{diag}(C(:,k))A$ denote the mapping matrix of cluster k. The decompressed gradients vector can be formulated as

$$\hat{g} = \sum_{k=1}^{K} g_k = \sum_{k=1}^{k} A_k ({A_k}^{\mathrm{T}} A_k)^{-1} A_k^{\mathrm{T}} g_k. \tag{2}$$

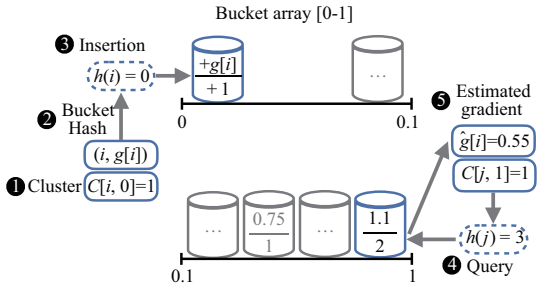

Figure 3 (Color online) The procedures of CASQ. The first three steps compress gradient $g(i)$ with the first bucket array. The last two steps decompress $g(j)$ from the second array.

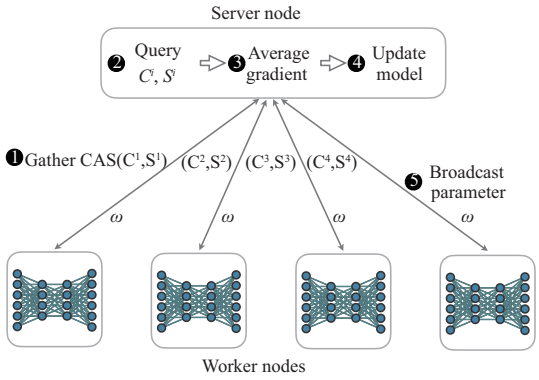

Figure 4 (Color online) The CASQ framework. The gradients are compressed with CAS and sent to the server (step 1). The server decompresses the gradients (step 2), aggregates them (step 3), and updates the model. Finally, the server broadcasts the model (step 5).

Adaptive bucket allocation. Another technology promoting the precision of CASQ is adaptive buckets allocation. Different gradient clusters require a different amount of buckets, as noted in Subsection 3.1. Hence, let m denote the total number of buckets for the sketch, and we configure the portion of each buckets array based on the two characteristics of a cluster. (1) Cluster entropy H: For a cluster covering small interval, the uncertainty of its gradient is at a low degree, a small portion of buckets is enough to achieve a low compression error. On the other hand, a cluster with a large interval requires a large number of buckets to handle its uncertainty. We quantify the uncertainty of cluster k as entropy $H_k = -\sum_{j \in S_k} f_j \log f_j$, where S_k denotes the set of gradients bins, and f_j denotes the binned values of relative frequency. (2) Cluster density μ: A large cluster center is in the tail of the gradient distribution. It does not require m to be too large to control the compression error. We quantify the density with the proportion of the cluster. Considering both of the factors, we allocate $m \times \frac{H_k \mu_k}{\sum H_i \mu_i}$ buckets for cluster k in the offline clustering algorithm.

3.3 Computing clustering assignment with K-means

CASQ uses the K-means clustering method to find the optimal quantization levels and classify each gradient.

When CASQ uses b bits to quantize a gradient, the number of quantization levels is $K = 2^b$. Thus the number of cluster centers is K since CASQ sets the cluster centers as the quantization levels to group gradients.

Let μ_k denote the expectation of kth cluster, $k \in [0, K-1]$. The objective of clustering in CASQ is to minimize the gradient classification error. This object can be represented as $\|g - \mathbb{E}[\hat{g}]\|_2^2$. Because we estimate the gradient with the average value within a bucket, we have $\mathbb{E}[\hat{g}] = \mu_k$. Thus,

$$\min \|g - \mathbb{E}[\hat{g}]\|_2^2 = \min \sum_{k=1}^{K} \sum_{j=1}^{N} C(j,k) \|g(j) - \mu_k\|_2^2, \tag{3}$$

where $C(j,k) = 1$ if $g(j)$ is assigned to the kth cluster, or $C(j,k) = 0$ otherwise.

Since the quantization levels (cluster centers) are μ_k in K-means, this objective function is equal to that of K-means. Thus, K-means can find the optimal clustering assignments.

3.4 The implementation of CASQ based SGD

The detail of our CASQ based SGD algorithm is summarized in Algorithm 1. We take the parameter server as an example, as depicted in Figure 4. After computing gradient with back-propagation, the algorithm first runs K-means to find K cluster centers (procedures `InitCASQ`). Due to the prohibitive cost of clustering on the whole gradient, we turn to learn the cluster centers on a sampled gradient vector and use them to assign total data. After clustering, we can also set the size of each bucket array in `InitCASQ`. Our method adaptively allocates buckets for each cluster based on the entropy and

Algorithm 1 Cluster-aware sketch quantization

Input: K: number of clusters; τ: clustering interval; W: number of work nodes; λ: momentum parameter; A: randomly mapping matrix;
Output: $\boldsymbol{\omega}_T$: model parameters;
Initialize: parameters $\boldsymbol{\omega}_0$, error compensation $e_0 \leftarrow 0$, momentum $\boldsymbol{m}_0 \leftarrow 0$, learning rate η_0;
1: **for all** $1 \leqslant i \leqslant W$ **do**
2: **for** $t \leftarrow 1, T$ **do**
3: Compute the stochastic gradient g_t^i;
4: **if** $t\%\tau == 0$ **then**
5: $\boldsymbol{c} \leftarrow$ Init_CASQ(g_t^i, K);
6: **end if**
7: $\tilde{g}_t^i \leftarrow g_t^i + e_{t-1}^i$; // Error compensation.
8: $S^i, C^i \leftarrow$ Compress$(g_t^i, \boldsymbol{c}, A)$;
9: $e_t^i = \tilde{g}_t^i -$ Decompress(S^i, C^i);
10: Gather(S^i, C^i);
11: **for all** S^i, C^i **do**
12: $\hat{g}_t^i \leftarrow$ Decompress(S^i, C^i);
13: **end for**
14: $\hat{g}_t \leftarrow \frac{1}{W}\sum_{i=1}^{W} \hat{g}_t^i$;
15: $\boldsymbol{m}_t \leftarrow \lambda \boldsymbol{m}_{t-1} + \hat{g}_t$;
16: $\boldsymbol{\omega}_t \leftarrow \boldsymbol{\omega}_{t-1} - \eta_t(\hat{g}_t + \boldsymbol{m}_t)$;
17: Broadcast$(\boldsymbol{\omega}_t)$;
18: **end for**
19: **end for**
20: **function** Compress$(g, \boldsymbol{c}, A)$
21: $C \leftarrow$ Assign_Cluster$(g, \boldsymbol{c})$;
22: **for All** $k \leftarrow 1, K$ **do**
23: $\Lambda[k] \leftarrow \mathrm{diag}(C[:, k])g$;
24: Counter$[k] \leftarrow \mathrm{diag}((A_k^{\mathrm{T}} A_k)^{-1})$;
25: **end for**
26: $S \leftarrow$ Counter $\odot\, (A^{\mathrm{T}}\Lambda)$; // Elementwise multiplication to compute the values of each bucket.
27: **return** S, C;
28: **end function**
29: **function** Decompress(S, C)
30: **for all** $k \leftarrow 1, K$ **do**
31: $\hat{g}[k] \leftarrow \mathrm{diag}(C[:, k])AS[k]$ // Getting the gradients of each cluster according to the cluster assignment C and mapping matrix A;
32: **end for**
33: $\hat{g} \leftarrow \sum_{k=1}^{K} \hat{g}_t^i[k]$;
34: **return** $\hat{g}$;
35: **end function**

center. Because the gradients distributions among consecutive iterations are almost the same (as shown in Figure 1(a)), procedure `InitCASQ` is executed every τ iterations in the implementation (lines 4–6), which makes the time cost of clustering negligible.

Next, gradients are quantized to C^i, and inserted to sketch S^i in worker node i (line 8). To efficiently assign each gradient to its nearest center, we parallelize the procedure `Assign_Cluster` with GPU.

Then, the compressed gradients are gathered by the server node, and decompressed as illustrated in Figure 3 (lines 10–13). Finally, the model parameters are updated with the decompressed gradient and broadcast to worker nodes (lines 14–17).

Note that because the decompressed gradient of CASQ is a biased estimation of the original value (see Theorem 1(1)), we apply error compensation [21] to ensure the convergence of CASQ (line 7).

Efficient implementation of CASQ is vital because its compression cost must be limited to the same level as other quantization methods. We leverage the multi-thread parallelism of GPU to accelerate our method. Our CASQ implementation provides two simple APIs to abstract the compression and decompression procedure. The `Compress` API takes the gradient vector and cluster centers as input and generates the compressed gradients as output. The gradient vector is compressed by the cluster assignment C and the hash buckets array S. We use the atomic operation to coordinate when multiple GPU threads encode C and insert gradients into S. The `Decompress` API unfolds each worker's CAS into the original gradients in parallel. We use the shared memory in GPU to accelerate the memory access when decoding C.

4 Theoretical analysis

In this section, we first quantify the compression error bound of CASQ. This bound indicates that CASQ

can converge in a single node. Then we prove its convergence in a distributed setting.

4.1 Compression error bound of CASQ

In Theorem 1, we firstly quantify the expectation and variance of the compression error of one gradient, based on the random mapping of each gradient. We prove that the variance is inversely proportional to the number of elements in its mapped bucket. Then, we bound the compression error and the second moment of the estimated gradient vector with $\|g\|_2^2$.

Theorem 1. Let $\{g_j^i\}$ denote the gradients inserted into the bucket corresponding to the jth gradient. Let $n_j = |\{g_j^i\}|$ denote the number of item inserted to the bucket corresponding to the jth gradient. Supposing that j belongs to cluster k, where gradients are i.i.d with expectation μ_k and variance σ_k, let N_k denote the number of elements in k. Then we have the following:

(1) $\mathbb{E}\left[\hat{g}(j) - g(j)\right] = \left(1 - \frac{m}{N_k}\right)\left(\mu_k - g(j)\right)$,

(2) $\mathrm{Var}\left[\hat{g}(j) - g(j)\right] = \frac{m(N_k-1)}{{N_k}^2}\left(\delta_k^2 + \left(1 - \frac{1}{m}\right)\mu_k^2\right)$,

(3) $\mathbb{E}\left[\|\hat{g}\|_2^2\right] \leqslant \|g\|_2^2$,

(4) $\mathbb{E}\left[\|\hat{g} - g\|_2^2\right] \leqslant (1-\delta)\|g\|^2$ for $\delta \in (0,1)$ if $m \ll N_k$ holds.

The full proof is deferred to Appendix A.1

4.2 Convergence of CASQ

The compression error bound of CASQ in Theorem 1 immediately implies that CASQ quantization is a δ-approximate compressor [21]. Then we can prove the convergence of CASQ in a single node with this property and extend the result to distributed CASQ based on Lemmas 1 and 2. The analysis shows that the CASQ converges at a rate matching that of vanilla distributed SGD under standard assumptions of smoothness of the loss function and the variance of gradients. The detailed proof is stated in Appendix A.4.

Assumption 1. In distributed setting with W worker nodes, let function $f : \mathbb{R}^d \to \mathbb{R}$ be L-smooth, and for any $\boldsymbol{\omega} \in \mathbb{R}^d$, both the local and global gradients are bounded, i.e., $\exists \sigma_i, \sigma \in \mathbb{R}$ such that $\|\nabla f_i(\boldsymbol{\omega})\| \leqslant \sigma_i, \|\nabla f(\boldsymbol{\omega})\| \leqslant \sigma$.

Definition 1 (δ-approximate compressor)**.** An operator $\mathcal{C}(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$ is a δ-approximate compressor if $\exists \delta \in (0,1]$ such that $\|\mathcal{C}(\boldsymbol{\omega}) - \boldsymbol{\omega}\|_2^2 \leqslant (1-\delta)\|\boldsymbol{\omega}\|_2^2$, $\forall \boldsymbol{\omega} \in \mathbb{R}^d$.

Lemma 1. Under Assumption 1, the error compensation e_t^i in worker node i is bounded for all iteration t in CASQ:

$$\left\|e_t^i\right\|^2 \leqslant \frac{4(1-\delta)}{\delta^2}\sigma_i^2.$$

Lemma 2. Under Assumption 1, given the learning rate $0 < \eta < 2/(\rho + L)$, $\rho > 0$, the loss function satisfies

$$f\left(\boldsymbol{\nu}_{t+1}\right) - f\left(\boldsymbol{\nu}_t\right) \leqslant \Delta_{\mathrm{CAS}}^t,$$

where $\boldsymbol{\nu}_t := \boldsymbol{\omega}_t - \eta \cdot \sum_{i=1}^W e_t^i$ and

$$\Delta_{\mathrm{CAS}}^t := -\eta\left(1 - \frac{(\rho+L)\eta}{2}\right)\left\|\nabla f\left(\boldsymbol{\omega}_t\right)\right\|^2 + \frac{L^2\eta^2}{2\rho}\left\|\sum_{i=1}^W e_t^i\right\|^2.$$

Theorem 2 (Non-convex convergence of CASQ)**.** Given $0 < \rho$, $0 < \eta < 2/(\rho + L)$, the CASQ outputs model parameters $\{\boldsymbol{\omega}_t\}_{t \geqslant 0}$ in T iterations. Under Assumption 1,

$$\min_{0 \leqslant t \leqslant T}\left\|\nabla f\left(\boldsymbol{\omega}_t\right)\right\|^2 \leqslant \frac{a\left(f^0 - f^*\right)}{\eta \cdot (T+1)} + b \cdot \eta,$$

where $f^0 := f\left(\boldsymbol{\omega}_0\right)$, f^* is the optimal solution, $a := \frac{2}{(2-(\rho+L)\eta)}$, $b := \frac{4L^2W\sigma^2(1-\delta)}{\rho\delta^2(2-(\rho+L)\eta)}$ and $\sigma^2 := \sum_{i=1}^W \sigma_i^2$.

Proof. Proof of sketch. Recall that we prove the compression error bound of CASQ in Theorem 1; it immediately implies that CASQ is a δ-approximate compressor. Then, we can prove the convergence of CASQ in a single node with this property [21], and extend the result to a distributed setting based on Lemmas 1 and 2.

Table 1 Validation accuracy on CIFAR-10/100 using 2/3-bit quantization (except for SuperSGD and TernGrad) with 4 GPUs

	ResNet32 CIFAR-10		ResNet32 CIFAR-100		ResNet56 CIFAR-100		ResNet110 CIFAR-100	
Method	2-bit	3-bit	2-bit	3-bit	2-bit	3-bit	2-bit	3-bit
D-SGD	**92.61±0.05**		**67.39±0.06**		**69.36±0.06**		**71.52±0.13**	
Qinf	87,14±0.03	89.90±0.02	60.32±0.08	64.22±0.14	62.81±0.06	66.45±0.06	63.29±0.12	67.75±0.08
TernGrad	89.78±0.09		64.19±0.08		65.35±0.06		66.13±0.14	
NUQSGD	65.19±0.05	82.36±0.09	–	–	35.10±0.09	54.04±0.08	–	–
AMQ	82.20±0.05	91.80±0.08	–	66.49±0.12	55.26±0.13	67.83±0.08	–	69.65±0.14
AMQ-N	83.15±0.06	91.70±0.05	–	66.66±0.12	55.50±0.06	67.70±0.03	–	70.01±0.12
ALQ	79.21±0.10	92.27±0.04	–	67.35±0.09	45.74±0.07	69.10±0.10	–	70.87±0.10
ALQ-N	80.80±0.05	92.47±0.03	–	66.78±0.12	–	69.22±0.10	–	70.83±0.06
CASQ	**92.49±0.04**	**92.88±0.05**	**66.57±0.11**	**67.47±0.06**	**68.86±0.05**	**69.24±0.04**	**71.09±0.13**	**71.29±0.10**

5 Evaluation

In this section, we seek to answer the following questions: Does CASQ reduce the gradient compression error and improve the convergence and accuracy of the model compared with the existing adaptive quantization method (Subsection 5.1)? How fast is CASQ compared with the state-of-the-art methods when distributed training various models on different scale GPU clusters (Subsection 5.2)? What is the impact of CASQ's design on the performance of CASQ, including K-means and adaptive bucket allocation (Subsections 5.4 and 5.3)? How robust is CASQ as the quantization bits and the size of sketch grows (Subsection 5.5)?

We have implemented CASQ with CUDA to quantize the gradient in parallel and deployed it in PyTorch. We train ResNet [6] on CIFAR-10/100 and ImageNet in up to 6 nodes. Each node has 4 NVIDIA GTX-2080 GPUs, and 10 Gbps Ethernet interconnect.

5.1 Gradient compression error and convergence

This subsection shows that our method can reduce the variance and improve the convergence compared with existing quantization methods. We compare CASQ to three kinds of baselines.

- Uniform quantization levels methods, including TernGrad (TRN) [13], and Qinf, which normalizes gradient with $\|g\|_\infty$ in QSGD [12].
- Non-uniform levels methods, including NUQSGD (NUQ) [22], ALQ, AMQ, ALQ-N and AMQ-N [14]. NUQSGD has exponential levels with exponential factor $p = 0.5$. ALQ-N and AMQ-N are the normalized variations of ALQ and AMQ.
- Vanilla data-parallel distributed SGD (D-SGD).

We train ResNet32 on CIFAR-10/100, and ResNet32/56/110 on CIFAR-100 in a 4-GPU node to demonstrate the superiority of our method in the convergence, especially when training with 2-bit quantization. The size of the mini-batch per GPU is 128, and the learning rate is 0.1, which decays at 40000 and 60000 steps. We test these methods with 2-bits (4 levels) and 3-bit (8 levels) quantization except for D-SGD and TernGrad. Some of the results are ignored because they fail to converge in our testing. The validation accuracy is listed in Table 1, the gradient variance and training loss are shown in Figures 5 and 6.

Compression error reduction. We evaluate the compression error with the average square error $\frac{\|\hat{g}-g\|_2^2}{N}$ of the compression methods. Although gradients of D-SGD are without compression, it also has the error because its gradients are stochastic with respect to the full-batch training. To compute the square error, we feed ten batches of data without updating the model every 100 training steps. We collect these stochastic gradients and compute the mean compression error for each gradient coordinate. In the setting of our experiment, CASQ is under 2-bit and 3-bit, and other methods use 3-bit quantization. The size of the sketch is 0.5% of the number of gradients. As shown in Figure 5, the compression error of CASQ-2bits is less than other 3-bit quantization methods. Before the 4×10^4 training steps, our method has almost the same curves as ALQ. However, after the learning rate decays, the gap between them enlarges. This shows that our method can adaptively reduce the compression error throughout the training process. CASQ is the closest to D-SGD, especially before the learning rate decays in the

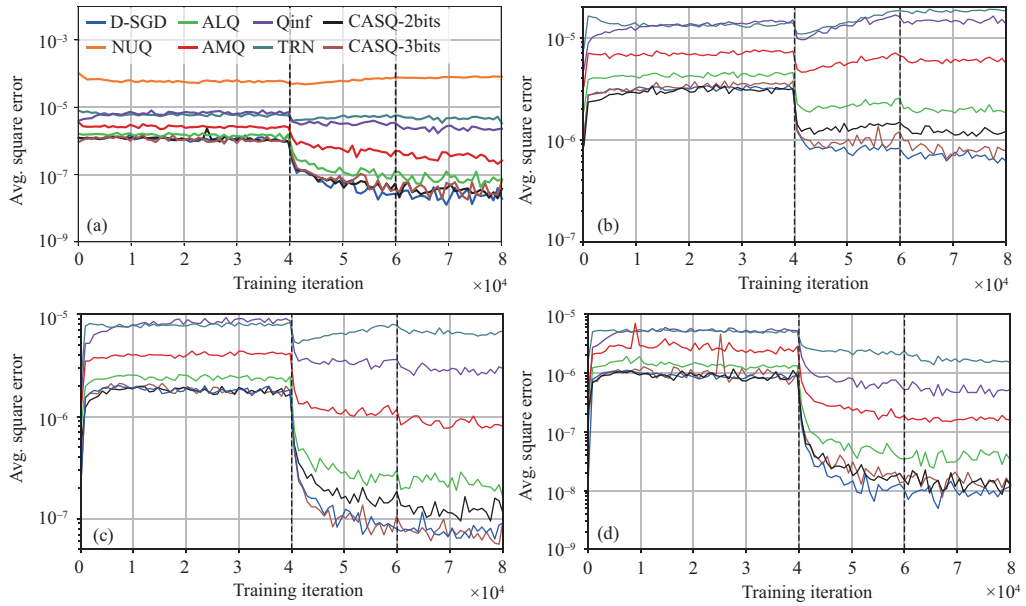

Figure 5 (Color online) Average square error of the compression methods. We train ResNet on CIFAR-10 and CIFAR-100. Our method uses 2 and 3 bits while others use 3 bits except for D-SGD and TernGrad. (a) ResNet32 on CIFAR-10; (b) ResNet32 on CIFAR-100; (c) ResNet56 on CIFAR-100; (d) ResNet110 on CIFAR-100.

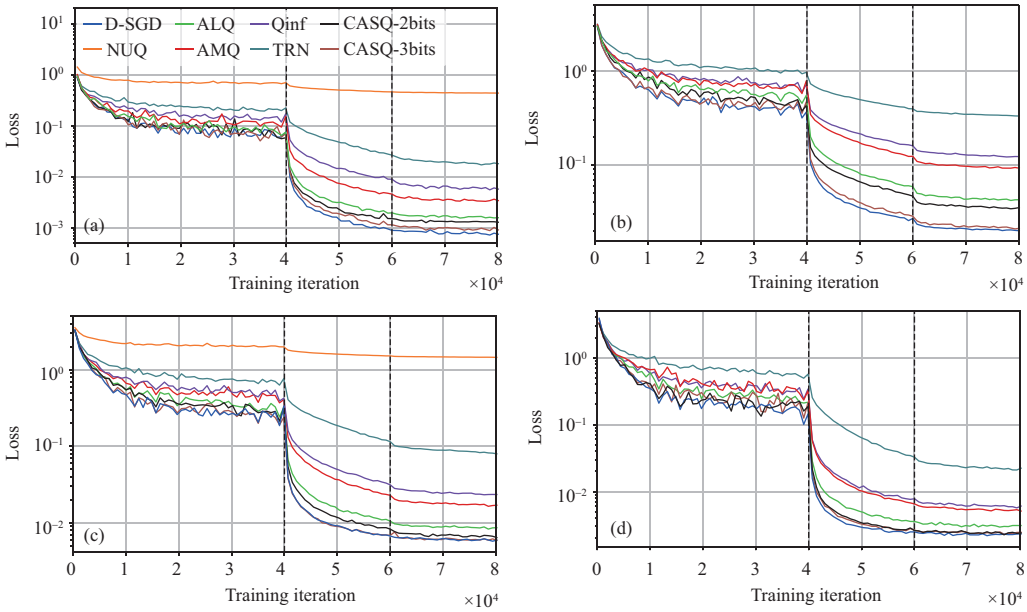

Figure 6 (Color online) Training loss of ResNet on CIFAR-10 and CIFAR-100. Our method uses 2 and 3 bits while others use 3 bits except for D-SGD and TernGrad. (a) ResNet32 on CIFAR-10; (b) ResNet32 on CIFAR-100; (c) ResNet56 on CIFAR-100; (d) ResNet110 on CIFAR-100.

40000 steps. Because the model parameter in this stage changes significantly, the proportion of large gradients is higher than that in the later stage. Thus, CASQ can reduce the compression error for more gradients. When we train ResNet56 on CIFAR-100, CASQ-3bits has almost the same curve as D-SGD, which means the compression error of CASQ-3bits reaches the same level of the stochastic gradients.

Improvement on convergence and accuracy. As illustrated in Figure 6, the loss curves of our method are the closest to that of distributed SGD and almost match the curves of SGD when quantization with 3 bits, which has not been achieved by any existing quantization studies. Moreover, the training loss of CASQ under 2-bit also converges faster than other methods that use 3-bit quantization. It means that CASQ can achieve a training target with lower communication costs in each iteration and requires fewer iterations. Our method also improves the validation accuracy. As shown in Table 1,

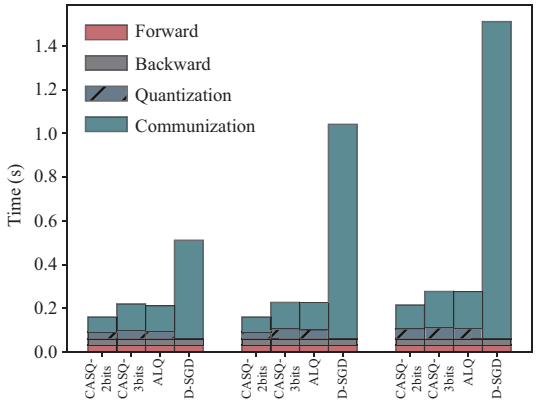

Figure 7 (Color online) The time cost of computation, quantization and communication in one iteration of CASQ, 3-bit ALQ and D-SGD when training ResNet110 on CIFAR-100 with (left) 8 workers, (middle) 16 workers, (right) 24 workers. The more workers, the more communication costs in each iteration.

the validation accuracy of our method is better than previous quantization methods. When training ResNet32 on CIFAR-10 with 2-bit quantization, our method leverages the accuracy by 2.7%. When training ResNet110 on CIFAR-100, our method also performs better than TernGrad by up to 4.9%, while other 2-bit adaptive quantization methods cannot converge. To achieve the same loss, CASQ-2bits requires at most 92% training steps compared with the 3-bit ALQ and allows more aggressive compression, which means our method can reduce the communication cost while maintaining the model performance. It is worth noting that when training on CIFAR-10 and CIFAR-100, we disable the error compensation, and our method also demonstrates its promised performance.

5.2 Performance of CASQ

Communication & quantization cost. The operations in our method are accelerated by GPU, which is the same as previous quantization methods. The time-consuming of compression and decompression is also at the same level as previous studies. In Figure 7, we compare the time-consuming of one training step of CASQ-2bits/3bits with D-SGD and 3-bit ALQ by training ResNet110 on CIFAR-100 with 8, 16 and 24 GPUs, 4 GPUs a node. Because the number of parameters of these models is small, we connect the nodes with 1 Gbps Ethernet interconnect. It shows that the quantization cost of CASQ is comparable to ALQ, which is up to 16% of the iteration time. In CASQ-2bits, the communication cost is less than 3-bit ALQ by up to 28%. In our method, the extra cost of updating the cluster centers is negligible. The experiments show that this part of costs is 0.5% of the iteration time at the most. Thus, with fewer training steps and iteration time, CASQ-2bits can accelerate distributed deep learning training.

The speedup of CASQ. As shown in Figure 6, the loss curves of CASQ-2bits/3bits decline faster than those existing quantization methods, which means it can reach a target accuracy in less time. To show the speedup of CASQ, we set the final loss of AMQ when training ResNet32/56/110 on CIFAR-100 as the target and compare the wall time of reaching this target loss in D-SGD and these previous studies. Each model is trained on 8 GPUs, 16 GPUs and 24 GPUs, as mentioned above. The training hyper-parameters are the same as in Subsection 5.1. As shown in Figure 8, our method achieves 5.4× speedup compared with D-SGD when training ResNet110 on 24 GPUs. Compared with ALQ, our method decreases up to 43% training time of ResNet110, up to 37% and 23% training time of ResNet56 and ResNet32, respectively, when training on an 8-GPU cluster.

We also quantify the improvement of end-to-end performance when training ResNet18 on ImageNet with 16 GPUs. The size of the mini-batch per GPU is 128. We initialize the learning rate with 0.1 and decay it at 30000 steps. As shown in Figure 9, CASQ-2bits reaches the same test accuracy as 3-bit ALQ after the learning rate decays and decreases 24% training time compared with 3-bit ALQ.

5.3 The effectiveness of adaptive bucket allocation

To show that our buckets allocation method can improve the convergence, we compare our method with an even buckets allocation CASQ, denoted by CASQ-W and CASQ-WO, respectively. We train ResNet32/56 with 2-bit quantization. The size of the sketch is 0.5% of the number of gradients. As

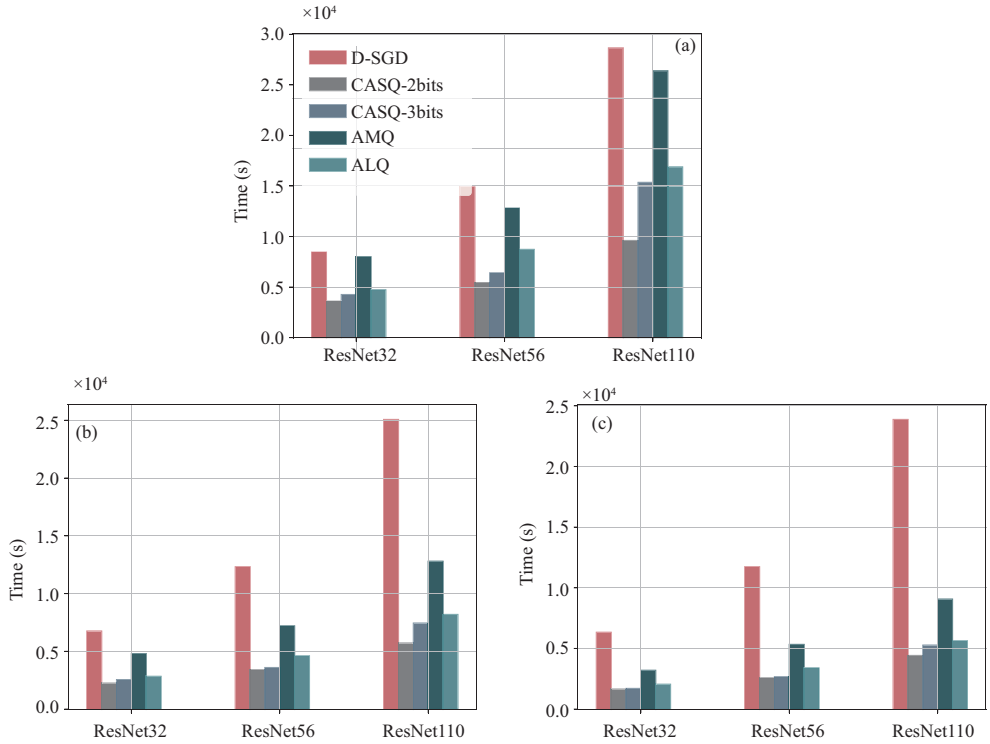

Figure 8 (Color online) The time-consuming and speedup of CASQ compared with D-SGD and existing quantization methods. (a) Training ResNet32/56/110 on CIFAR-100 with 8 GPUs; (b) training ResNet32/56/110 on CIFAR-100 with 16 GPUs; (c) training ResNet32/56/110 on CIFAR-100 with 24 GPUs.

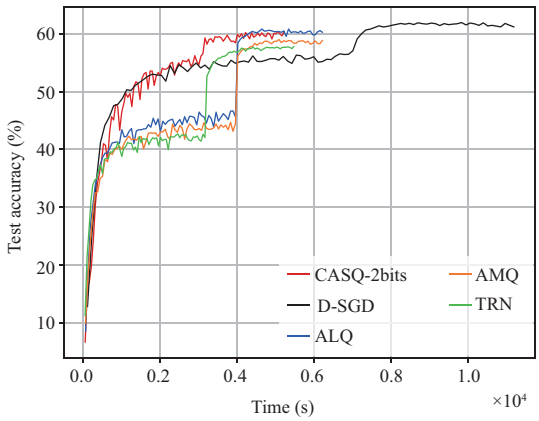

Figure 9 (Color online) The time-consuming of CASQ-2bits compared with D-SGD and existing 3-bit quantization methods when training ResNet18 on ImageNet with 16 GPUs.

illustrated in Figure 10, the convergence of CASQ-W is faster by 1.8×10^4 steps. It indicates that our buckets allocation method can reduce the training time to converge to a target loss.

5.4 The effectiveness of clustering

The optimal quantization levels. To validate the effectiveness of computing quantization levels with K-means, we substitute the clustering centers with the values from ALQ to quantize the gradient and then combine it with the sketch. This method is denoted by ALQ-Sketch. We train ResNet56 on CIFAR-100 with 2/3-bit ALQ-Sketch and CASQ-2bits/3bits. As shown in Figure 11, the accuracy declines over 5%, and the convergence deteriorates significantly, even worse than the native ALQ method. It is because the quantization levels computing from ALQ are not the optimal points to minimize the compression error of gradient estimated with the sketch, and K-means can find these values.

The cost of clustering. Our experiments balance the overhead of K-means and the model accuracy

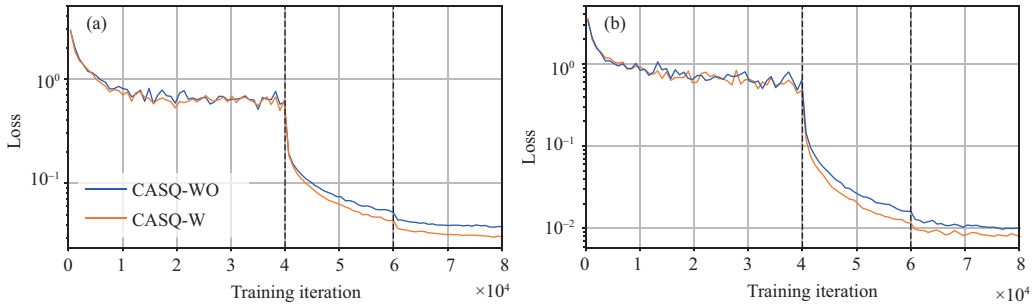

Figure 10 (Color online) Training loss CASQ with/without adaptive bucket allocation. (a) ResNet32 on CIFAR-100; (b) ResNet56 on CIFAR-100.

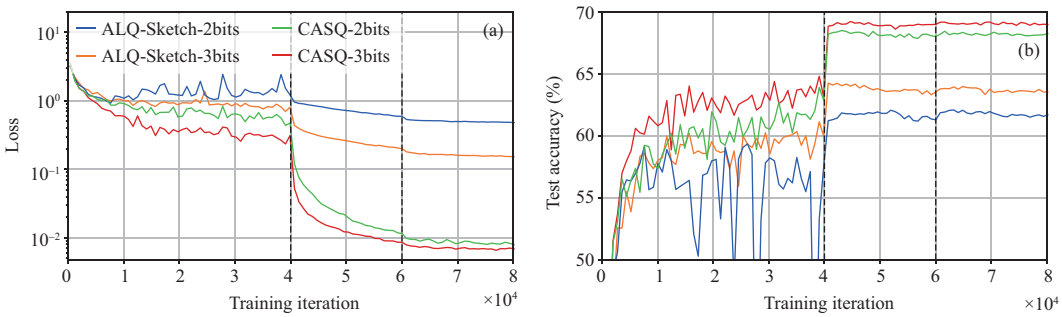

Figure 11 (Color online) Training loss and validation accuracy of ALQ-Sketch compared with CASQ when training ResNet56 on CIFAR-100. (a) Training loss; (b) validation accuracy.

by clustering on the sampled 10% gradients. The hyper-parameter τ is 100. We empirically show the effect of sampling in Figure 12. When CASQ samples less than 10% gradients, the overhead of K-means accounts for less than 0.5% of the entire training time. However, the test accuracy is downgraded by nearly 1%. When clustering on the whole gradient vector, it accounts for nearly 3% of the entire training time. We sample 10% gradients because it maintains the model accuracy and costs a negligible clustering overhead (0.5% of the entire time).

5.5 Sensitivity

Number of bits. In Figure 13, we study the robustness of our method to the quantization bits. We train ResNet32/56 on CIFAR-100, respectively, with 1-bit to 4-bit quantization.

Compared with quantization CASQ-3bits, the 2-bit quantization does not deteriorate the performance. The accuracy declines by less than 1%, while some other quantization studies cannot converge in our evaluation. When quantization with 4 bits, the results do not show better performance. This result indicates that 3 bits are good enough to obtain model quality comparable to the full-precision D-SGD.

The size of sketch. The performance of CASQ is robust to the size of the sketch. We tune the total number of buckets from 0 to 2% of gradients. For the method with no bucket, we store the mean of each cluster as the estimated value for their gradient, denoted by CASQ-no-Sketch. As shown in Figure 14, increasing the size of sketch array improves the convergence slightly after the size of the sketch reaches 0.2%. Thus, our method can get the promised performance with negligible extra communication cost of the sketch.

6 Related work

Communication efficient distributed training. The scalability of a distributed deep learning system is limited by the gradient synchronization time [23], and many recent communication efficient systems have been proposed to overlap the cost [24–31]. On the other hand, gradient compression algorithms also draw researchers' attention to reduce communication traffic. The current gradient compression methods can be categorized into three types: quantization, sparsification, and low-rank approximation. Quantization is a widely used gradient compression method because of its solid convergence guarantee [12, 13, 21, 32].

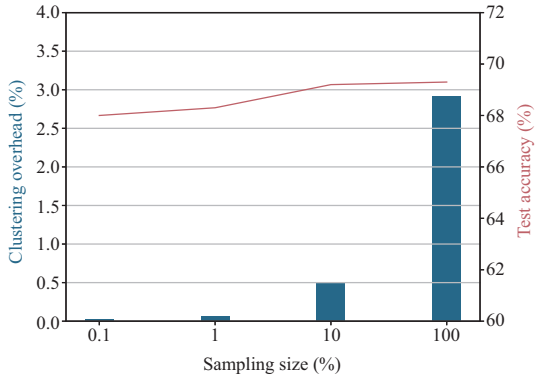

Figure 12 (Color online) The proportion of time spent on clustering and test accuracy. We train ResNet56 on CIFAR-100 with 16 GPUs. The sampling size varies from 0.1% to 100% (without sampling).

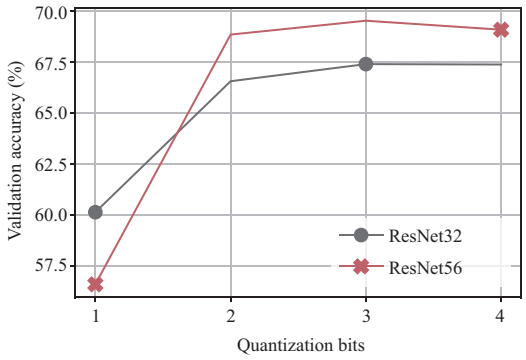

Figure 13 (Color online) Validation accuracy of CASQ when varying the number of quantization bits.

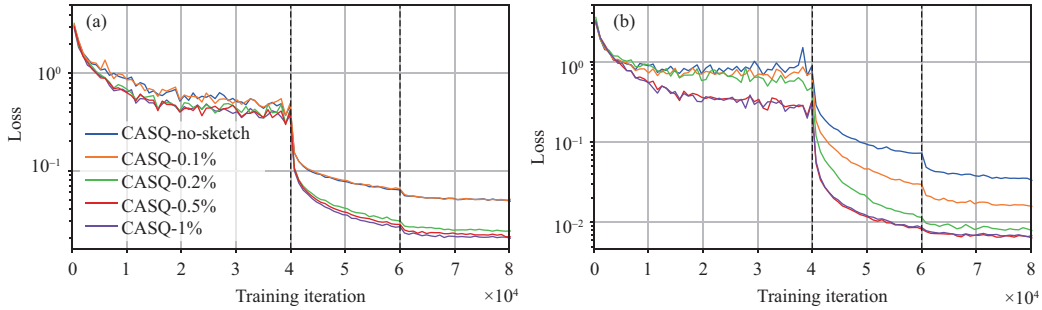

Figure 14 (Color online) Training loss of CASQ when varying the number of buckets. (a) ResNet32 with CASQ-3bits; (b) ResNet56 with CASQ-2bits.

The remarkable studies on quantization, such as QSGD [12] normalize and quantize each gradient with a set of uniform quantization levels, then represent the gradients with fewer bits. TernGrad [13] is a special case of QSGD, which only has three quantization levels. Besides, another line of compression methods aggressively sparsifies the gradient and only sends those significant [33–38]. Also, some recent studies use singular value decomposition to compress the gradient [16, 39, 40], which enables the linear computation on the compressed gradient. In this paper, we focus on the quantization method.

Adaptive quantization. Some prior attempts pay attention to the quantization of the non-uniform distributed gradients. ZipML [41] dynamically finds the optimal quantization levels, but it is difficult to quantize gradients on the fly. NUQSGD [42] and natural compression (NC) [43] quantize gradient with exponentially spaced levels. TINYSCRIPT [44] finds the quantization levels with minimum compression error based on modeling gradients with Weibull distribution. Faghri et al. [14] proposed ALQ and AMQ. ALQ minimizes the compression error of quantization by computing the optimal levels on the estimated gradient distribution. AMQ models the quantization levels as exponentially spaced levels and solves the optimal multiplier of the levels. However, the compression error reduction of these adaptive methods is limited by the discrete quantization levels. Recent work [45] also used the clustering method to minimize the compression error. However, similarly with ALQ/AMQ, it does not address the problem that the large gradient usually leads to significant compression error.

Sketch-based gradient compression. Sketch-based methods are introduced in large-scale training by Sketch-SGD [46] and in federated learning by FetchSGD [47]. However, the errors of these methods are bounded with L^2 norm. Thus they can only approximate the Top-K gradients. Prior study [48] used quantile sketch and Min-Max sketch to approximate the quantiles of gradient, but it focuses on linear models and is ill-suited to GPU-based deep learning systems.

7 Conclusion

This paper proposes CASQ, a novel sketch-based gradient quantization method. CASQ minimizes the compression error on the non-uniformly distributed gradient by dynamically clustering and adaptively allocating buckets for different clusters to compress the gradient. Experimental results show that CASQ effectively reduces the gradient compression error compared with existing methods. We implement CASQ in the large-scale deep learning system. We show that CASQ-2bits convergence is faster and improves the accuracy of models with negligible communication cost. With fewer training iterations and more aggressive gradient compression, CASQ can decrease the training time. CASQ benefits the training of deep learning models in the large-scale GPU clusters and workloads in edge computing such as federated learning.

Acknowledgements This work was supported in part by National Natural Science Foundation of China (Grant Nos. 62025208, 61972409) and National Key Research Development Program of China (Grant No. 2021YFB0301200).

References

1 Robbins H, Monro S. A stochastic approximation method. Ann Math Statist, 1951, 22: 400–407
2 Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learning Res, 2011, 12: 2121–2159
3 Sutskever I, Martens J, Dahl G, et al. On the importance of initialization and momentum in deep learning. In: Proceedings of International Conference on Machine Learning, 2013. 1139–1147
4 Ben-Nun T, Hoefler T. Demystifying parallel and distributed deep learning. ACM Comput Surv, 2020, 52: 1–43
5 Dean J, Corrado G, Monga R, et al. Large scale distributed deep networks. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2012. 1223–1231
6 He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. 770–778
7 Mahajan D, Girshick R, Ramanathan V, et al. Exploring the limits of weakly supervised pretraining. In: Proceedings of European Conference on Computer Vision, 2018. 181–196
8 Devlin J, Chang M W, Lee K, et al. BERT: pre-training of deep bidirectional transformers for language understanding. 2018. ArXiv:1810.04805
9 Tan M, Le Q. EfficientNet: rethinking model scaling for convolutional neural networks. In: Proceedings of International Conference on Machine Learning, 2019. 6105–6114
10 Carion N, Massa F, Synnaeve G, et al. End-to-end object detection with transformers. In: Proceedings of European Conference on Computer Vision, 2020. 213–229
11 Xie Q, Luong M T, Hovy E, et al. Self-training with noisy student improves ImageNet classification. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020. 10687–10698
12 Alistarh D, Grubic D, Li J, et al. QSGD: communication-efficient SGD via gradient quantization and encoding. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2017. 1709–1720
13 Wen W, Xu C, Yan F, et al. TernGrad: ternary gradients to reduce communication in distributed deep learning. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2017. 1509–1519
14 Faghri F, Tabrizian I, Markov I, et al. Adaptive gradient quantization for data-parallel SGD. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, 2020. 33: 3174–3185
15 Johnson R, Zhang T. Accelerating stochastic gradient descent using predictive variance reduction. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2013. 315–323
16 Wang H, Sievert S, Liu S, et al. ATOMO: communication-efficient learning via atomic sparsification. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2018. 9850–9861
17 Li M, Andersen D G, Park J W, et al. Scaling distributed machine learning with the parameter server. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, 2014. 583–598
18 Cormode G, Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications. J Algorithms, 2005, 55: 58–75
19 Cormode G, Hadjieleftheriou M. Finding the frequent items in streams of data. Commun ACM, 2009, 52: 97–105
20 Fu Y, Li D, Shen S, et al. Clustering-preserving network flow sketching. In: Proceedings of IEEE INFOCOM 2020, 2020. 1309–1318
21 Karimireddy S P, Rebjock Q, Stich S, et al. Error feedback fixes SIGNSGD and other gradient compression schemes. In: Proceedings of International Conference on Machine Learning, 2019. 3252–3261
22 Xu H, Ho C Y, Abdelmoniem A M, et al. GRACE: a compressed communication framework for distributed machine learning. In: Proceedings of 2021 IEEE 41st International Conference on Distributed Computing Systems, 2021. 561–572
23 You Y, Buluç A, Demmel J. Scaling deep learning on GPU and knights landing clusters. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, New York, 2017
24 Abadi M, Barham P, Chen J, et al. TensorFlow: a system for large-scale machine learning. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, 2016. 265–283
25 Zhang H, Zheng Z, Xu S, et al. Poseidon: an efficient communication architecture for distributed deep learning on GPU clusters. In: Proceedings of 2017 USENIX Annual Technical Conference (USENIX ATC 17), 2017. 181–193
26 Li Y, Yu M, Li S, et al. Pipe-SGD: a decentralized pipelined SGD framework for distributed deep net training. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018. 8056–8067
27 Huang Y, Cheng Y, Bapna A, et al. GPipe: efficient training of giant neural networks using pipeline parallelism. 2018. ArXiv:1811.06965
28 Shi S, Chu X, Li B. MG-WFBP: efficient data communication for distributed synchronous SGD algorithms. In: Proceedings of IEEE INFOCOM 2019-IEEE Conference on Computer Communications, 2019. 172–180
29 Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: generalized pipeline parallelism for DNN training. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019. 1–15

30 Rasley J, Rajbhandari S, Ruwase O, et al. DeepSpeed: system optimizations enable training deep learning models with over 100 billion parameters. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020. 3505–3506
31 Fan S, Rong Y, Meng C, et al. DAPPLE: a pipelined data parallel approach for training large models. In: Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2021. 431–445
32 Bernstein J, Wang Y X, Azizzadenesheli K, et al. SIGNSGD: compressed optimisation for non-convex problems. In: Proceedings of International Conference on Machine Learning, 2018. 560–569
33 Stich S U, Cordonnier J B, Jaggi M. Sparsified SGD with memory. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2018. 4447–4458
34 Lin Y, Han S, Mao H, et al. Deep gradient compression: reducing the communication bandwidth for distributed training. In: Proceedings of International Conference on Learning Representations, 2018
35 Wangni J, Wang J, Liu J, et al. Gradient sparsification for communication-efficient distributed optimization. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2018. 1299–1309
36 Renggli C, Ashkboos S, Aghagolzadeh M, et al. SparCML: high-performance sparse communication for machine learning. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, 2019. 1–15
37 Shi S, Zhao K, Wang Q, et al. A convergence analysis of distributed SGD with communication-efficient gradient sparsification. In: Proceedings of International Joint Conference on Artificial Intelligence, 2019. 3411–3417
38 Chen C Y, Ni J, Lu S, et al. ScaleCom: scalable sparsified gradient compression for communication-efficient distributed training. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2020
39 Yu M, Lin Z, Narra K, et al. GradiVeQ: vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2018. 5123–5133
40 Vogels T, Karimireddy S P, Jaggi M. PowerSGD: practical low-rank gradient compression for distributed optimization. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2019. 14259–14268
41 Zhang H, Li J, Kara K, et al. ZipML: training linear models with end-to-end low precision, and a little bit of deep learning. In: Proceedings of International Conference on Machine Learning, 2017. 4035–4043
42 Ramezani-Kebrya A, Faghri F, Roy D M. NUQSGD: improved communication efficiency for data-parallel SGD via nonuniform quantization. 2019. ArXiv:1908.06077
43 Horvath S, Ho C Y, Horvath L, et al. Natural compression for distributed deep learning. 2019. ArXiv:1905.10988
44 Fu F, Hu Y, He Y, et al. Don't waste your bits! Squeeze activations and gradients for deep neural networks via tinyscript. In: Proceedings of International Conference on Machine Learning, 2020. 3304–3314
45 Cui L, Su X, Zhou Y, et al. Slashing communication traffic in federated learning by transmitting clustered model updates. IEEE J Sel Areas Commun, 2021, 39: 2572–2589
46 Ivkin N, Rothchild D, Ullah E, et al. Communication-efficient distributed SGD with sketching. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2019. 13144–13154
47 Rothchild D, Panda A, Ullah E, et al. FetchSGD: communication-efficient federated learning with sketching. In: Proceedings of International Conference on Machine Learning, 2020. 8253–8265
48 Jiang J, Fu F, Yang T, et al. SKCompress: compressing sparse and nonuniform gradient in distributed machine learning. VLDB J, 2020, 29: 945–972

Appendix A Proof of lemmas and theorems

Appendix A.1 Proof of Theorem 1

Proof. The error of a sketch can be quantified as $\hat{g}-g=AA^{\mathrm{T}}g-g=(AA^{\mathrm{T}}-I)g$, where I denotes the identity matrix. AA^{T} is a $N\times N$ symmetric matrix, where each entry (i,j) is one when the ith and jth gradients mapped into the same bucket, i.e., $A(i,:)=A(j,:)$, and zero otherwise. It means that the diagonal entries are all set to ones. Each gradient is mapped according to a uniformly-random hash function, so that $\Pr[AA^{\mathrm{T}}(i,j)=1]=\frac{1}{m}$ and $\Pr[AA^{\mathrm{T}}(i,j)=0]=1-\frac{1}{m}$, which follow the Bernoulli distribution.

Let $\Phi=AA^{\mathrm{T}}-I$, then each non-diagonal entry in Φ is the same as that in AA^{T}, we have $\Phi_{i\neq j}(i,j)\sim\text{Bernoulli}(\frac{1}{m})$, and the diagonal entries are zeros, which means $\mathbb{E}[\Phi_{i\neq j}(i,j)]=\frac{1}{m}$ and $\text{Var}[\Phi_{i\neq j}(i,j)]=\frac{1}{m}\cdot(1-\frac{1}{m})$.

Note that a gradient is randomly mapped to the buckets, then n_j is viewed as a constant $\frac{N_k}{m}$. The approximated $g(j)$ is the averaged bucket value, so that the expectation of loss is

$$\begin{aligned}\mathbb{E}[\hat{g}(j)-g(j)]&=\mathbb{E}\left[\frac{\sum g_j^i}{n_j}-g(j)\right]=\frac{1}{n_j}\mathbb{E}\left[\sum_{i=1}^{n_j-1}\left(g_j^i-g(j)\right)\right]\\&=\frac{(n_j-1)\left(\mu_k-g(j)\right)}{n_j}=\left(1-\frac{m}{N_k}\right)\left(\mu_k-g(j)\right)\\&\leqslant\frac{(N-1)\mu}{m}.\end{aligned}$$

Recall that we derive the gradients from (2), which has the similar structure form with $AA^{\mathrm{T}}g$ for each cluster, and the random variable in one cluster is independent of that in other clusters. Therefore, we note the variance of $\hat{g}(j)-g(j)$ as below:

$$\begin{aligned}\text{Var}\left[\hat{g}(j)-g(j)\right]&=\text{Var}\left[\sum_{i=1}^{N_k}\left(A\left(A^{\mathrm{T}}A\right)^{-1}A^{\mathrm{T}}\right)(i,j)g(i)-g(j)\right]\\&=\text{Var}\left[\frac{\sum_{i=1}^{N_k}\left(AA^{\mathrm{T}}\right)(i,j)g(i)}{n_j}-g(j)\right]\\&=\frac{1}{n_j^2}\text{Var}\left[\sum_{i=1}^{N_k}\left(AA^{\mathrm{T}}\right)(i,j)g(i)-n_jg(j)\right]\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{n_j^2}\mathrm{Var}\left[\sum_{i=1}^{N_k-1}\Phi\left(i,j\right)g(i) - (n_j-1)g(j)\right]\\
&= \frac{N_k-1}{n_j^2}\mathrm{Var}[\Phi(i,j)g(i)]\\
&= \frac{N_k-1}{n_j^2}\cdot\frac{1}{m}\left(\delta_k^2+\left(1-\frac{1}{m}\right)\mu_k^2\right)\\
&= \frac{m(N_k-1)}{{N_k}^2}\left(\delta_k^2+\left(1-\frac{1}{m}\right)\mu_k^2\right).
\end{aligned}
$$

To prove the bound of $\mathbb{E}\left[\|\hat{g}\|_2^2\right]$, we note the following:

$$
\mathbb{E}\left[\hat{g}(j)^2\right] = \mathbb{E}\left[\left(\frac{\sum_{i=1}^{n_j} g_j^i}{n_j}\right)^2\right] \leqslant \mathbb{E}\left[\frac{\sum_{i=1}^{n_j}\left(g_j^i\right)^2}{n_j}\right],
$$

where the second line follows the fact that the arithmetic mean is less or equal than the quadratic mean. Hence, we have

$$
\begin{aligned}
\mathbb{E}\left[\|\hat{g}\|_2^2\right] &= \sum_{j=1}^{N}\mathbb{E}\left[(\hat{g}(j))^2\right] = \sum_{k=1}^{K}\sum_{j=1}^{N_k}\mathbb{E}\left[\left(\frac{\sum_{i=1}^{n_j} g_j^i}{n_j}\right)^2\right]\\
&\leqslant \sum_{k=1}^{K}\sum_{j=1}^{N_k}\mathbb{E}\left[\frac{\sum_{i=1}^{n_j}\left(g_j^i\right)^2}{n_j}\right] = \sum_{k=1}^{K}\sum_{j=1}^{N_k} g(j)^2\\
&= \|g\|_2^2.
\end{aligned}
$$

Based on the lemma above, we prove the bound of $\mathbb{E}\left[\|\hat{g}-g\|_2^2\right]$ as follows:

$$
\begin{aligned}
\mathbb{E}\left[\|\hat{g}-g\|_2^2\right] &= \sum_{k=1}^{K}\sum_{j=1}^{N_k}\mathbb{E}\left[(\hat{g}(j)-g(j))^2\right] = \sum_{k=1}^{K}\sum_{j=1}^{N_k}\mathrm{Var}\left[\hat{g}(j)-g(j)\right] + \mathbb{E}\left[\hat{g}(j)-g(j)\right]^2\\
&= \sum_{k=1}^{K}\sum_{j=1}^{N_k}\frac{N_k-1}{n_j^2}\cdot\frac{1}{m}\left(\delta_k^2+\left(1-\frac{1}{m}\right)\mu_k^2\right) + \left(\frac{(n_j-1)\left(\mu_k-g(j)\right)}{n_j}\right)^2\\
&\leqslant \sum_{k=1}^{K}\sum_{j=1}^{N_k}\frac{N_k}{n_j^2}\cdot\frac{1}{m}\left(\delta_k^2+\left(1-\frac{1}{m}\right)\mu_k^2\right) + \left(\mu_k-g(j)\right)^2\\
&= \sum_{k=1}^{K}\sum_{j=1}^{N_k}\frac{m}{N_k}\left(\delta_k^2+\left(1-\frac{1}{m}\right)\mu_k^2\right) + \mu_k^2 - 2\mu_k g(j) + g(j)^2\\
&= \sum_{k=1}^{K} m\left(\delta_k^2+\left(1-\frac{1}{m}\right)\mu_k^2\right) + N_k\mu_k^2 - 2\mu_k\|g_k\|_1 + \|g_k\|_2^2\\
&\overset{(a)}{\approx} \sum_{k=1}^{K}\mu_k\left(N_k\mu_k - 2\|g_k\|_1\right) + \|g_k\|_2^2\\
&\leqslant \sum_{k=1}^{K}\|g_k\|_2^2 = \|g\|_2^2,
\end{aligned}
$$

where (a) follows from the setting of sketch that $m \ll N_k$. As a result, we have $\mathbb{E}\left[\|\hat{g}-g\|_2^2\right] \leqslant (1-\delta)\|g\|^2$ for $\delta \in (0,1)$, as claimed.

Appendix A.2 Proof of Lemma 1

Proof. As shown in Definition 1, $\mathcal{C}(\cdot)$ is a δ-approximate compressor. Combined with the definition of the error sequence, we have

$$
\begin{aligned}
\left\|e_{t+1}^i\right\|_2^2 = \left\|g_t^i - \mathcal{C}(g_t^i)\right\|_2^2 &\leqslant (1-\delta)\|g_t^i\|_2^2\\
&= (1-\delta)\|\nabla f_i(\omega_t) + e_t^i\|_2^2\\
&\leqslant (1-\delta)\left[(1+\gamma)\|e_t^i\|_2^2 + (1+1/\gamma)\|\nabla f_i(\omega_t)\|_2^2\right],
\end{aligned}
$$

where we use the Young's inequality (for any $\gamma > 0$). Noticed that $e_0^i = 0$ and $\nabla f_i(\omega_t)$ is bounded in Assumption 1, simple computation yields

$$\begin{aligned}\left\|e_{t+1}^i\right\|_2^2 &\leqslant (1-\delta)(1+\gamma)\|e_t^i\|_2^2 + (1-\delta)(1+1/\gamma)\|\nabla f_i(\omega_t)\|_2^2\\ &\leqslant (1-\delta)(1+\gamma)^{t+1}\|e_0^i\|_2^2 + \sum_{k=0}^{t}\left[(1-\delta)(1+\gamma)\right]^{t-k}(1-\delta)(1+1/\gamma)\|\nabla f_i(\omega_t)\|_2^2\\ &\leqslant \sum_{k=0}^{\infty}\left[(1-\delta)(1+\gamma)\right]^{t-k}(1-\delta)(1+1/\gamma)\sigma_i^2\\ &= \frac{(1-\delta)(1+1/\gamma)}{1-(1-\delta)(1+\gamma)}\sigma_i^2.\end{aligned}$$

Let us pick $\gamma = \frac{\delta}{2(1-\delta)}$ such that $1+1/\gamma = (2-\delta)/\delta \leqslant 2/\delta$. Then we have

$$\left\|e_{t+1}^i\right\|_2^2 \leqslant \frac{2(1-\delta)(1+1/\gamma)}{\delta}\sigma_i^2 \leqslant \frac{4(1-\delta)}{\delta^2}\sigma_i^2.$$

Appendix A.3 Proof of Lemma 2

Proof. From the definition of the sequence $\{\nu_t\}_{t=0,1,\ldots}$, we have that

$$\begin{aligned}\nu_{t+1} &= \omega_{t+1} - \eta\sum_{i=1}^{W}\boldsymbol{e}_{t+1}^i = \omega_t - \eta\sum_{i=1}^{W}\mathcal{C}(g_t^i) - \eta\sum_{i=1}^{W}\boldsymbol{e}_{t+1}^i\\ &= \omega_t - \eta\sum_{i=1}^{W}g_t^i = \omega_t - \eta\sum_{i=1}^{W}\boldsymbol{e}_t^i - \eta\sum_{i=1}^{W}\nabla f_i(\omega_t)\\ &= \nu_t - \eta\nabla f(\omega_t).\end{aligned}$$

Since the function f is L-smooth, i.e.,

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leqslant L\|\boldsymbol{x}-\boldsymbol{y}\|_2, \quad f(\boldsymbol{x}) - f(\boldsymbol{y}) \leqslant \langle\nabla f(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle + \frac{L}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2.$$

We have

$$\begin{aligned}f(\nu_{t+1}) - f(\nu_t) &\leqslant \langle\nabla f(\nu_t), \nu_{t+1}-\nu_t\rangle + \frac{L}{2}\|\nu_{t+1}-\nu_t\|_2^2\\ &\leqslant \langle\nabla f(\omega_t), \nu_{t+1}-\nu_t\rangle + \langle\nabla f(\nu_t) - f(\omega_t), \nu_{t+1}-\nu_t\rangle + \frac{L}{2}\|\nu_{t+1}-\nu_t\|_2^2\\ &\leqslant -\eta\left(1-\frac{L\eta}{2}\right)\|\nabla f(\omega_t)\|_2^2 + \frac{\rho}{2}\|\nu_{t+1}-\nu_t\|_2^2 + \frac{1}{2\rho}\|\nabla f(\nu_t) - f(\omega_t)\|_2^2\\ &\leqslant -\eta\left(1-\frac{(L+\rho)\eta}{2}\right)\|\nabla f(\omega_t)\|_2^2 + \frac{L^2}{2\rho}\|\nu_t-\omega_t\|_2^2\\ &\leqslant -\eta\left(1-\frac{(\rho+L)\eta}{2}\right)\|\nabla f(\omega_t)\|_2^2 + \frac{L^2\eta^2}{2\rho}\left\|\sum_{i=1}^{W}\boldsymbol{e}_t^i\right\|_2^2.\end{aligned} \tag{A1}$$

The third inequality follows from the mean-value inequality and holds for any $\rho > 0$.

Appendix A.4 Proof of Theorem 2

Proof. The one-step descent of CASQ is given in Lemma 2, and the residual error is bounded in Lemma 1. Summing the terms in (A1) over t gives

$$f(\nu_{T+1}) - f(\nu_0) \leqslant -\eta\left(1-\frac{(\rho+L)\eta}{2}\right)\sum_{t=0}^{T}\|\nabla f(\omega_t)\|_2^2 + \frac{2L^2\eta^2(1-\delta)}{\rho\delta^2}(T+1)W\sigma^2, \tag{A2}$$

where we use

$$\left\|\sum_{i=1}^{W}\boldsymbol{e}_t^i\right\|_2^2 \leqslant W\sum_{i=1}^{W}\|\boldsymbol{e}_t^i\|_2^2 \leqslant W\cdot\frac{4(1-\delta)}{\delta^2}\sum_{i=1}^{W}\sigma_i^2 := \frac{4(1-\delta)}{\delta^2}W\sigma^2,$$

where $\sigma^2 := \sum_{i=1}^{W}\sigma_i^2$.

Notice that $\nu_0 = \omega_0$ and f^* is the optimal value. Given step-size $\eta < 2/(\rho+L)$, rearranging the terms in (A2) and averaging over t lead to

$$\begin{aligned}\frac{1}{T+1}\sum_{t=0}^{T}\|\nabla f(\omega_t)\|_2^2 &\leqslant \frac{f(\nu_0) - f(\nu_{T+1})}{\eta(T+1)\left(1-\frac{(\rho+L)\eta}{2}\right)} + \frac{2L^2\eta(1-\delta)W\sigma^2}{\rho\delta^2\left(1-\frac{(\rho+L)\eta}{2}\right)}\\ &\leqslant \frac{2(f^0-f^*)}{\eta(T+1)\left(2-(\rho+L)\eta\right)} + \frac{4L^2\eta(1-\delta)W\sigma^2}{\rho\delta^2\left(2-(\rho+L)\eta\right)}.\end{aligned}$$