Pipelined Gradient Coding

Xian Su
Department of Computer Science
Florida International University
Miami, FL, USA
Email: xsu@fiu.edu

Jun Li
Queens College and The Graduate Center
City University of New York
New York, NY, USA
Email: jun.li@qc.cuny.edu

***Abstract*—In large-scale machine learning, distributed training commonly involves multiple workers evaluating the gradients of the model on different dataset partitions. A common challenge is the presence of straggling workers, which may significantly slow down training. Traditional gradient coding (GC) addresses this by duplicating dataset partitions across workers, allowing for the replacement of missing gradients from stragglers. However, GC requires workers to evaluate gradients on multiple dataset partitions in each step, potentially increasing overall training time. In this paper, we propose to pipeline GC, such that gradient evaluation is segmented across multiple steps and each worker evaluates gradients on just a single dataset partition per step. We develop the pipelined version for fractional repetition (FR) and cyclic repetition (CR), two representative dataset placement schemes in GC, and prove convergence guarantees for both. Through extensive simulations and experiments on cloud infrastructure, our schemes not only significantly reduce training time but also accelerate convergence compared to GC and other baselines.**

I. Introduction

The training of machine-learning models typically involves solving an optimization problem where model parameters are iteratively updated using gradients evaluated over a dataset. With modern models and datasets growing rapidly, it is common practice to train in parallel on multiple *workers*, where each worker evaluates gradients over a single dataset partition, and results are aggregated on a *master* to update the model [1]–[6].

During distributed training over many workers, it is common to encounter *stragglers*, *i.e.*, workers performing more slowly than others, which can significantly delay training, as each step requires gradients from all dataset partitions [6]–[8].

Gradient coding (GC) [6] was proposed to tolerate up to s stragglers by assigning $c = s+1$ dataset partitions per worker so that the aggregated gradient can be recovered from any $n-s$ workers. However, GC requires workers to evaluate gradients on c partitions per step, which can take c times longer [9]. If the straggler delay is less than the extra computation time, overall training time may increase rather than decrease. In Sec. III, we empirically show this can occur in practice.

In this paper, we propose pipelined gradient coding (PGC)[1], which mitigates stragglers without the additional computational workload of GC. PGC breaks multi-partition gradient computation into multiple steps in a pipelined fashion: each worker evaluates gradients on just one dataset partition per step and uses stale gradients from previous steps to form coded gradients. We design PGC for both fractional repetition (FR) and cyclic repetition (CR), prove convergence guarantees for both variants, and demonstrate through extensive experiments that PGC significantly reduces training time and improves convergence.[2]

II. Related Work

Replication-based straggler mitigation assigns the same task to multiple workers [11]–[13]. While simple, replication requires prohibitively many extra workers. Coding-based methods [4], [5], [14]–[17] encode dataset partitions for matrix multiplication tasks, enabling recovery from a subset of workers. However, these apply only to linear operations.

GC [6] applies coding to gradients, making it applicable to general gradient-based models. Extensions include Reed-Solomon-based constructions to minimize expected computation time [18] and communication-computation tradeoffs [19]. Partial gradient recovery schemes [17], [20]–[22] reduce time by exploiting stragglers' partial results, but all still require each worker to evaluate gradients on c partitions per step [9]. Most closely related, Sequential GC [23] also exploits the temporal dimension, but through multi-round scheduling with relaxed per-task deadlines and adaptive reattempts across rounds, while each worker still evaluates c partitions per task. In contrast, PGC pipelines the per-partition computation *within a single* training process, reusing stale gradients so that each worker evaluates only one partition per step.

Approximate recovery methods tolerate stragglers by recovering partial gradients. IS-SGD [24]–[26] simply discards straggler gradients. Approximate GC (AGC) and stochastic GC (SGC) [17], [27]–[29] recover more gradients than IS-SGD but sacrifice recovery accuracy and are limited to specific parameter combinations. IS-GC [30] maximizes recovered gradients in GC with an arbitrary number of stragglers. However, these methods still subject workers to the additional c-partition

[1]This work also appears in the first author's Ph.D. dissertation [10].

[2]An extended version with complete proofs of all results is available at https://arxiv.org/abs/2607.20739.

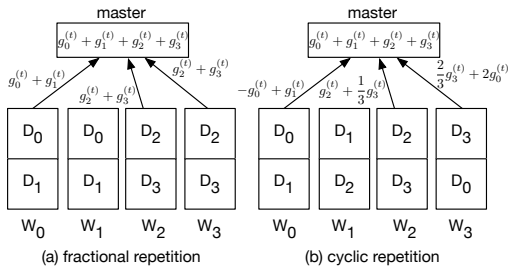

Fig. 1: Illustrative examples of FR and CR, both with $n = 4$ and $c = 2$.

workload (except IS-SGD). PGC reduces the per-worker workload to the same level as DGD while approximately recovering the full aggregated gradients.

III. Preliminary and Motivation

Consider a dataset D with d samples and p features. We aim to find optimal parameters via $\beta^* = \arg\min_{\beta\in\mathbb{R}^p} F(\beta; D) = \arg\min_{\beta\in\mathbb{R}^p} \frac{1}{d}\sum_{i=0}^{d-1} f(\beta; x_i, y_i)$. In the t-th step, $\beta^{(t+1)} = \beta^{(t)} - \eta g^{(t)}$, where η is the learning rate. For distributed gradient descent (DGD) with n workers, $g^{(t)} = \frac{1}{n}\sum_{i=0}^{n-1} g_i^{(t)}$, where $g_i^{(t)}$ denotes the gradients on partition D_i at step t.

GC recovers $\sum_{i=0}^{n-1} g_i^{(t)}$ from any $n-s$ workers by assigning each worker $c = s+1$ dataset partitions. Two representative placement schemes are fractional repetition (FR) and cyclic repetition (CR), illustrated in Fig. 1.

FR requires $c|n$, dividing n workers into $\frac{n}{c}$ groups. Worker W_i holds partitions $D_{jc}, \ldots, D_{jc+(c-1)}$, where $j = \lfloor \frac{i}{c} \rfloor$. Each worker uploads the sum of its gradients; since workers in the same group send identical coded gradients, any one worker per group suffices, tolerating up to $s = c-1$ stragglers.

CR does not require $c|n$, placing partitions cyclically: W_i holds $\{D_{j \bmod n} | j = i, \ldots, i+c-1\}$. Each worker uploads a linear combination of its c partition gradients; the master applies straggler-pattern-dependent decoding coefficients to the $n-s$ received coded gradients to recover $\sum_i g_i^{(t)}$ exactly [6]. This offers greater flexibility than FR at the cost of a more involved coding construction.

However, GC requires each worker to evaluate gradients on its $c = s+1$ partitions per step, multiplying per-step computation by c. Fig. 2 shows that this overhead causes GC to consistently take more time per step than DGD ($> 100\%$), with larger s making it worse. Unless stragglers are sufficiently frequent and slow, the extra computation cost of GC outweighs the time saved from straggler mitigation, leaving total training time no better, or even worse, than DGD.

Fig. 3 illustrates how PGC addresses this. Rather than evaluating c partitions in one step, each worker rotates through its assigned partitions one per step, retaining the most recent gradient from each. At each step, it assembles a coded gradient by combining the fresh gradient with $c-1$ retained stale values, replicating the GC coded gradient structure; the master then collects responses from the $n-s$ fastest workers and decodes

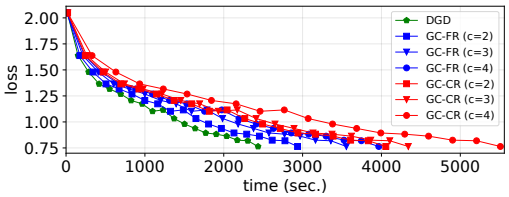

Fig. 2: Training time per step normalized by DGD when training ResNet-18 on CIFAR-10 with $n = 12$ workers on Google Cloud, showing that GC consistently requires more time than DGD due to its c-fold computational overhead.

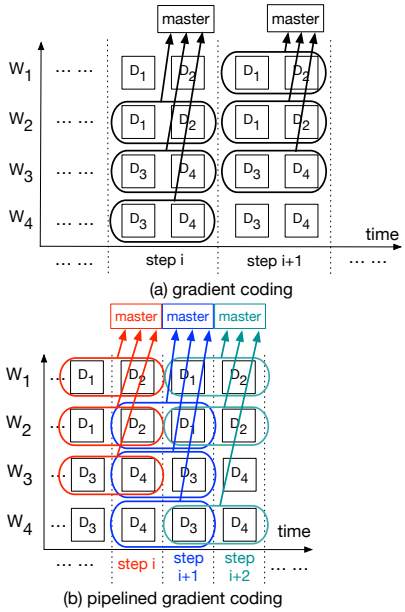

Fig. 3: A comparison between gradient coding and pipelined gradient coding.

as in GC. Each worker still uploads exactly one coded gradient per step, so PGC's per-step communication is identical to GC and DGD and incurs no additional communication overhead, with only the local computation pipelined. The cost is that $c-1$ component gradients are stale, introducing a bounded approximation error that diminishes with the learning rate. Sections IV and V detail the pipelined encoding and decoding of PGC for FR and CR, respectively, and analyze their convergence, establishing that PGC converges geometrically under standard assumptions, matching GC's convergence structure while reducing per-step computation to that of DGD.

IV. Pipelined Gradient Coding with Fractional Repetition

In this section, we present the design of PGC for FR. The design for CR is presented in Section V.

A. Coding Scheme

With FR, W_i holds c partitions $\{D_{jc}, \ldots, D_{jc+(c-1)}\}$ where $j = \lfloor \frac{i}{c} \rfloor$. In GC, worker W_i uploads the coded gradient

$\sum_{l=0}^{c-1} g_{jc+l}^{(t)}$, requiring its c partitions to be evaluated at every step.

PGC-FR reduces this to one partition per step. Worker W_i rotates cyclically through its c partitions: at step t, it evaluates $D_{jc+((t-1) \bmod c)}$ and forms the coded gradient $\hat{g}_i^{(t)} = \sum_{l=0}^{c-1} g_{jc+((t-1-l) \bmod c)}^{(t-l)}$ from the most recent evaluation of each partition. This coded gradient coincides with the GC coded gradient when all evaluations are current and approximates it when staleness is small.

The master decodes in the same way as GC: select one worker per group from the received $n-s$ coded gradients and aggregate across groups. An initial step $t=0$ evaluates its c partitions to warm up the pipeline.

B. Convergence Analysis

We now analyze the convergence of PGC-FR. Let $s(t,i) \in [t-c+1, t]$ denote the step at which dataset partition D_i's gradient contributing to the update at step t was evaluated. The update rule is $\beta^{(t+1)} = \beta^{(t)} - \eta \hat{g}^{(t)}$, where $\hat{g}^{(t)} = \frac{1}{n} \sum_{i=0}^{n-1} g_i^{(s(t,i))}$.

Theorem 1: Assume that $F(\beta)$ is λ-strongly convex and $\nabla F(\beta)$ is L-Lipschitz continuous. Assume the gradient variance is bounded: $\mathbb{E}[||g_i^{(s(t,i))} - \nabla F(\beta^{(s(t,i))})||_2^2] \leq \xi \mathbb{E}[||\nabla F(\beta^{(s(t,i))})||_2^2]$ for some $\xi \in [0,1)$. Assume staleness is bounded: $\mathbb{E}[||\nabla F(\beta^{(t)}) - \nabla F(\beta^{(s(t,i))})||_2^2] \leq \gamma \mathbb{E}[||\nabla F(\beta^{(t)})||_2^2]$ for some $\gamma \in [0,1)$. Then with learning rate $\eta \leq \frac{1}{nL(n+\xi)}$:

$$\mathbb{E}[F(\beta^{(t)})] - F(\beta^*) \leq \left(1 - \frac{\lambda(1-\gamma)}{nL(n+\xi)}\right)^t (F(\beta^{(0)}) - F(\beta^*)).$$

Theorem 1 guarantees geometric convergence under standard assumptions. The staleness is inherently bounded by the pipeline depth ($s(t,i) \in [t-c+1, t]$), so γ is governed by c and the learning rate η, vanishing as $\eta \to 0$.

Proof sketch. Starting from the Lipschitz inequality and the polarization identity $\mathbf{a}^T\mathbf{b} = \frac{1}{2}||\mathbf{a}||^2 + \frac{1}{2}||\mathbf{b}||^2 - \frac{1}{2}||\mathbf{a}-\mathbf{b}||^2$, taking expectations yields:

$$\begin{aligned}\mathbb{E}[F(\beta^{(t+1)})] \leq \mathbb{E}[F(\beta^{(t)})] - \tfrac{\eta(1-\gamma)}{2}\mathbb{E}[||\nabla F(\beta^{(t)})||_2^2] \\ - \tfrac{\eta}{2n}(1 - L\eta n(n+\xi)) \\ \cdot \textstyle\sum_{i=0}^{n-1} \mathbb{E}[||\nabla F(\beta^{(s(t,i))})||_2^2],\end{aligned}$$

where the staleness bound γ controls the inner-product error term, and the variance bound ξ gives $\mathbb{E}[||\hat{g}^{(t)}||_2^2] \leq \frac{n+\xi}{n} \sum_i \mathbb{E}[||\nabla F(\beta^{(s(t,i))})||_2^2]$. With $\eta = \frac{1}{nL(n+\xi)}$ the last sum is non-negative, and applying λ-strong convexity $||\nabla F(\beta)||_2^2 \geq 2\lambda(F(\beta) - F(\beta^*))$ yields the stated contraction. The full proof is given in the extended version.

V. Pipelined Gradient Coding with Cyclic Repetition

A. Motivation

Similar to PGC-FR, our initial attempt to build PGC-CR was to directly apply the pipelining principle to CR using the coding scheme from GC. This attempt, however, failed. As shown in Fig. 4, this naïve design does not converge, while PGC-FR under the same training configuration converges stably.

The failure stems from how GC-CR decodes. Recovery of $\sum_i g_i^{(t)}$ requires the master to solve a linear system over the $n-s$ received coded gradients, yielding straggler-pattern-dependent *decoding coefficients* $\alpha_{m,k}$ for worker m's contribution to partition k. These satisfy $\sum_m \alpha_{m,k} = 1$ for each k, ensuring exact recovery when gradients are current, but the individual $\alpha_{m,k}$ are unconstrained in magnitude: for poorly conditioned straggler patterns they reach the thousands, up to ± 7500 for $n=12$, $c=6$. In GC-FR, by contrast, each partition is contributed by one worker with coefficient 1.

With PGC's stale gradients, each contribution to partition k carries a staleness error $\delta_{m,k}$. In PGC-FR it enters with coefficient 1 and stays small; in naïve PGC-CR each $\delta_{m,k}$ is scaled by $|\alpha_{m,k}|$, and the errors do not cancel despite $\sum_m \alpha_{m,k} = 1$, amplifying the net error by up to $7500\times$. Fig. 4 plots the max/min decoding coefficients $\alpha_{m,k}$ with the training loss: the naïve PGC-CR loss spikes precisely when they spike.

B. Coding Scheme

We therefore design PGC-CR to avoid large decoding coefficients. Rather than requiring exact recovery of the full gradient sum, we relax to a *weighted* sum whose expected coefficient for each partition equals 1, *i.e.*, $\mathbb{E}\left[\frac{1}{\mu} \middle| N_k^{(t)} \middle|\right] = 1$, where $\mu = \frac{(n-s)c}{n}$ and $|N_k^{(t)}|$ is the total number of times a gradient on partition D_k appears across the $n-s$ received coded gradients at step t.

With $D_{(j+l) \bmod n}$ on W_j, $l = 0, \ldots, c-1$, worker W_j evaluates $D_{(j+(t-1) \bmod c) \bmod n}$ at step t and forms the coded gradient $\hat{g}_j^{(t)} = \sum_{l=0}^{c-1} g_{(j+(t-1-l) \bmod c) \bmod n}^{(t-l)}$ from the most recent c evaluations, all with coefficient 1. The master collects coded gradients from the $n-s$ fastest workers and computes $\hat{g}^{(t)} = \frac{1}{n\mu} \sum_{m=0}^{n-s-1} \hat{g}_{j_m}^{(t)}$, where the additional factor $\frac{1}{n}$ rescales the recovered weighted sum to match DGD's averaged gradient. In expectation over the straggler pattern, $\hat{g}^{(t)} = \frac{1}{nc} \sum_{i=0}^{n-1} \sum_{j=0}^{c-1} g_i^{(s(t,i,j))}$, consistent with the estimator analyzed in Theorem 2. This design ensures $0 \leq \frac{1}{\mu}|N_k^{(t)}| < 2$, bounding the amplification of stale gradient errors.

C. Convergence Analysis

We now analyze the convergence of PGC-CR. Let $s(t,i,j) \in [t-c+1, t]$ denote the step at which the gradient on D_i evaluated by worker $W_{(i+j) \bmod n}$ and used in the update at step t was computed, for $0 \leq j \leq c-1$; $s(t,i,j) < t$ when that gradient is stale. With the all-ones coefficient design, $\mathbb{E}[\hat{g}^{(t)}] = \frac{1}{nc} \sum_{i=0}^{n-1} \sum_{j=0}^{c-1} g_i^{(s(t,i,j))}$. Let $\bar{\nabla}^{(t)} = \frac{1}{nc} \sum_{i=0}^{n-1} \sum_{j=0}^{c-1} \nabla F(\beta^{(s(t,i,j))})$ and $\bar{g}^{(t)} = \frac{1}{nc} \sum_{i=0}^{n-1} \sum_{j=0}^{c-1} g_i^{(s(t,i,j))}$. The proof of Theorem 2 rests on two lemmas.

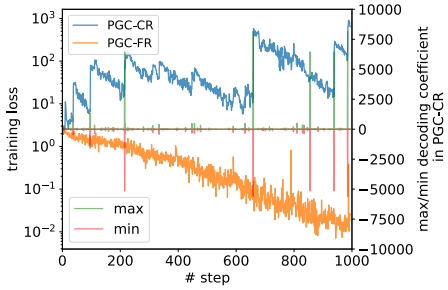

Fig. 4: Training with PGC-FR and PGC-CR for $n = 12$, $c = 6$.

Lemma 1: Assume $\mathbb{E}[g_i^{(t)}] = \nabla F(\beta^{(t)})$ for all i. Then:

$$\mathbb{E}[||\nabla F(\beta^{(t)}) - \hat{g}^{(t)}||_2^2] = \mathbb{E}[||\nabla F(\beta^{(t)}) - \bar{\nabla}^{(t)}||_2^2] - \mathbb{E}[||\bar{\nabla}^{(t)}||_2^2] + \mathbb{E}[||\bar{g}^{(t)}||_2^2].$$

Lemma 1 decomposes the gradient estimation error into a staleness bias term ($\mathbb{E}[||\nabla F(\beta^{(t)}) - \bar{\nabla}^{(t)}||_2^2]$, controlled by γ) and a stochastic variance term ($\mathbb{E}[||\bar{g}^{(t)}||_2^2]$, controlled by ξ), with a negative cross term that tightens the bound. The proof adds and subtracts $\bar{\nabla}^{(t)}$ and uses unbiasedness to cancel cross terms.

Lemma 2: Assume $\mathbb{E}[||g_i^{(s(t,i,j))} - \nabla F(\beta^{(s(t,i,j))})||_2^2] \leq \xi \mathbb{E}[||\nabla F(\beta^{(s(t,i,j))})||_2^2]$ for some $\xi \in [0, 1)$. Then:

$$\mathbb{E}\left[\left\|\sum_{i=0}^{n-1}\sum_{j=0}^{c-1} g_i^{(s(t,i,j))}\right\|_2^2\right] \leq (nc + \xi)\sum_{i=0}^{n-1}\sum_{j=0}^{c-1}\mathbb{E}\left[\left\|\nabla F(\beta^{(s(t,i,j))})\right\|_2^2\right].$$

Lemma 2 bounds the squared norm of the total gradient sum by $(nc + \xi)$ times the sum of squared individual norms. The factor nc arises because, for any two distinct evaluations $(i_1, j_1) \neq (i_2, j_2)$, the cross term $\mathbb{E}[(g_{i_1}^{(s)} - \nabla F(\beta^{(s)}))^\top (g_{i_2}^{(s)} - \nabla F(\beta^{(s)}))]$ vanishes: each stochastic gradient is an unbiased, conditionally independent estimate given its evaluation point, so the expectation factorizes to zero. Only the nc diagonal terms survive, and the variance bound ξ controls each residual. The full derivation is given in the extended version.

Theorem 2: Assume that $F(\beta)$ is λ-strongly convex and $\nabla F(\beta)$ is L-Lipschitz continuous. Assume that for some $\xi \in [0, 1)$, $\mathbb{E}[||g_i^{(s(t,i,j))} - \nabla F(\beta^{(s(t,i,j))})||_2^2] \leq \xi \mathbb{E}[||\nabla F(\beta^{(s(t,i,j))})||_2^2]$. Define $\bar{\nabla}^{(t)} = \frac{1}{nc}\sum_{i=0}^{n-1}\sum_{j=0}^{c-1}\nabla F(\beta^{(s(t,i,j))})$. Assume that for some $\gamma \in [0, 1)$: $\mathbb{E}\left[\left|\left|\nabla F(\beta^{(t)}) - \bar{\nabla}^{(t)}\right|\right|_2^2\right] \leq \gamma \mathbb{E}[||\nabla F(\beta^{(t)})||_2^2]$. Then with learning rate $\eta \leq \frac{1}{L(nc+\xi)}$:

$$\mathbb{E}[F(\beta^{(t)})] - F(\beta^*) \leq \left(1 - \frac{\lambda(1-\gamma)}{L(nc+\xi)}\right)^t (F(\beta^{(0)}) - F(\beta^*)).$$

Proof sketch. Starting from the Lipschitz inequality and the polarization identity, taking expectations and applying Lemma 1 yields:

$$\mathbb{E}[F(\beta^{(t+1)})] \leq \mathbb{E}[F(\beta^{(t)})] - \frac{\eta(1-\gamma)}{2}\mathbb{E}[||\nabla F(\beta^{(t)})||_2^2] - \frac{\eta}{2}\mathbb{E}[||\bar{\nabla}^{(t)}||_2^2] + \frac{L\eta^2}{2}\mathbb{E}[||\hat{g}^{(t)}||_2^2],$$

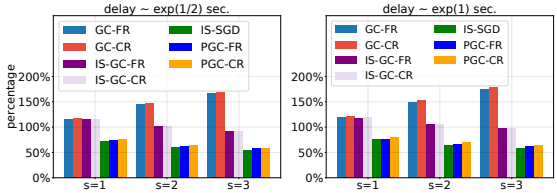

Fig. 5: Time per step when training ResNet-18 on ImageNet, normalized by DGD (100%). PGC achieves stable near-DGD per-step time regardless of s.

where the staleness bound γ controls the first-order bias via Lemma 1. Writing $\hat{g}^{(t)} = \frac{1}{nc}\sum_{i,j} g_i^{(s(t,i,j))}$ and applying Lemma 2 gives:

$$\mathbb{E}[||\hat{g}^{(t)}||_2^2] \leq \frac{nc+\xi}{n^2c^2}\sum_{i,j}\mathbb{E}[||\nabla F(\beta^{(s(t,i,j))})||_2^2].$$

With $\eta \leq \frac{1}{L(nc+\xi)}$, the positive term is dominated by $-\frac{\eta}{2}\mathbb{E}[||\bar{\nabla}^{(t)}||_2^2]$, yielding $\mathbb{E}[F(\beta^{(t+1)})] \leq \mathbb{E}[F(\beta^{(t)})] - \frac{\eta(1-\gamma)}{2}\mathbb{E}[||\nabla F(\beta^{(t)})||_2^2]$; λ-strong convexity then gives the stated contraction. The full proof of Lemmas 1–2 and Theorem 2 is given in the extended version.

Unlike PGC-FR, where each partition's gradient comes from one worker, PGC-CR averages gradients from different workers with varying staleness; this mixing reduces the adverse effect of staleness and speeds up convergence in practice.

VI. Evaluation

We implement PGC using Ray [31] for distributed computing and PyTorch for training. Each worker evaluates gradients on one dataset partition per step and keeps gradients from the previous $c-1$ steps for encoding. The master uses Ray.wait() to collect coded gradients from the $n-s$ fastest workers and decodes using the respective FR or CR scheme. The optimizer is torch.optim.SGD.

We compare PGC against DGD, GC (with FR and CR), IS-SGD, and IS-GC [30]. IS-SGD ignores straggler gradients; IS-GC combines GC with straggler ignoring. All implementations use identical random seeds to ensure fair comparison.

A. Simulation

We first evaluate through simulations on a local high-performance computing (HPC) cluster with $n = 12$ workers, training ResNet-18 on ImageNet [32]. To simulate stragglers, we add independent random delays on each worker following exponential distributions with means of 1 and 2 seconds, respectively. Results are averaged over 5 runs; training runs for 25 epochs.

Time per step. Fig. 5 compares per-step time across $s \in \{1, 2, 3\}$. GC (FR and CR) consistently exceeds 100% due to the c-partition computation per step, worsening with s. IS-GC reduces overhead at higher s via a fixed $c = 2$. PGC achieves stable, near-DGD per-step time for all s, since each worker evaluates exactly one partition per step while tolerating up to s stragglers.

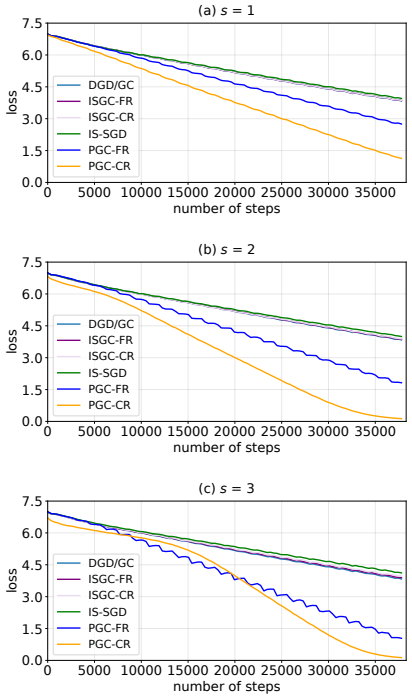

Fig. 6: Training loss over steps for ResNet-18 on ImageNet with $s = 1, 2, 3$. GC/DGD curves coincide; PGC converges faster with the advantage growing for larger s.

Convergence. Fig. 6 shows training loss over 25 epochs. GC coincides with DGD since both recover the same aggregated gradient. IS-SGD and IS-GC converge at rates comparable to DGD: IS-SGD discards straggler gradients, reducing the effective batch size and degrading convergence; IS-GC avoids this bias but its c-fold per-step computation offsets the straggler-tolerance benefit. PGC converges noticeably faster, with the advantage growing as s increases. In particular, PGC-CR consistently outperforms PGC-FR due to CR's mixed staleness (elaborated in Sec. VI-B).

B. Experiments in the Cloud

We train ResNet-18 [33] on CIFAR-10 [34] on a Google Cloud cluster of $n = 12$ workers (n2-highmem-2) and one master (e2-highmem-2), with $s \in \{1, 2, 3\}$. No artificial delays are added; natural stragglers arise from the cloud environment. Training runs until loss reaches 0.5; each scheme is run 10 times. Batch size is 256 and learning rate is 0.01.

Training time. Fig. 7 shows loss over training time. GC takes the longest for each s due to higher per-step computation. Regardless of s, PGC reaches the loss threshold fastest, with PGC-CR outperforming PGC-FR in all cases.

Time per step. Fig. 8a compares average time per step, normalized to DGD (100%). GC consumes more time than DGD due to its computational overhead, while IS-GC costs less at higher s. Since PGC evaluates only one partition per step, it achieves almost the same time per step as IS-SGD.

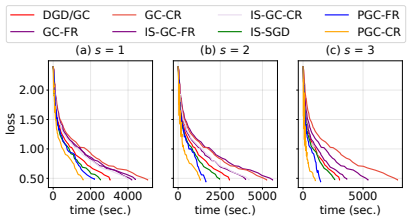

Fig. 7: Loss reduction over time when training ResNet-18 on CIFAR-10 using DGD, GC, IS-GC, IS-SGD, and PGC for $s = 1, 2, 3$.

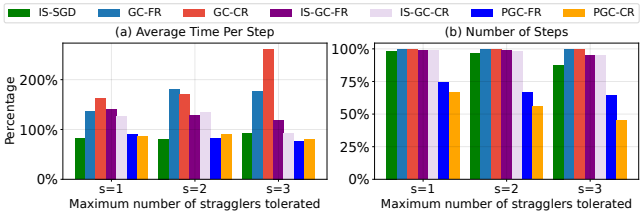

Fig. 8: Time consumption training ResNet-18 on CIFAR-10, normalized by DGD (100%).

Convergence. Fig. 8b shows steps to reach the loss threshold. We still use DGD as the baseline (100%). GC always matches DGD as it recovers the same aggregated gradients. PGC achieves significantly fewer steps, up to 54.85% fewer than DGD. For IS-GC, we fix $c = 2$ for all s, the minimum for straggler ignoring, so its per-step computation is twice DGD's; despite this, IS-GC beats DGD only at $s = 3$, since the tolerated-straggler fraction must exceed the extra-computation fraction to yield a gain. IS-SGD instead discards straggler gradients, avoiding the overhead but degrading convergence through a downward bias on the effective batch size. PGC uses one partition per step (like IS-SGD) while recovering coded gradients that approximate the full aggregated gradient (like GC), attaining the best of both.

As in simulations, PGC-CR converges faster than PGC-FR, mirroring our analysis: the contraction factor $\left(1 - \frac{\lambda(1-\gamma)}{L(nc+\xi)}\right)$ in Theorem 2 is smaller than $\left(1 - \frac{\lambda(1-\gamma)}{nL(n+\xi)}\right)$ in Theorem 1 whenever $c \leq n$. Intuitively, CR's "mixed" staleness, *i.e.*, gradients on the same partition coming from different workers with different delays, averages out the adverse effect of stale gradients, akin to variance reduction in asynchronous optimization [25]. Combining better per-step time and faster convergence, PGC substantially reduces total training time.

VII. Conclusion

We proposed pipelined gradient coding (PGC), which overcomes the c-fold computation overhead of gradient coding by pipelining: each worker evaluates only one partition per step and reuses stale gradients to form coded gradients. We proved convergence for both FR and CR variants and showed that PGC matches DGD's efficiency while outperforming prior schemes in training time.

REFERENCES

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal Distributed Online Prediction Using Mini-Batches," *Journal of Machine Learning Research*, vol. 13, no. 1, 2012.

[3] M. Zinkevich, M. Weimer, L. Li, and A. Smola, "Parallelized stochastic gradient descent," *Advances in neural information processing systems*, vol. 23, 2010.

[4] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication," *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, 2017.

[5] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding Up Distributed Machine Learning Using Codes," in *IEEE Transactions on Information Theory*, vol. 64, no. 3, 2018, pp. 1514–1529.

[6] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 70, 2017, pp. 3368–3376.

[7] J. Dean and L. A. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[8] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[9] E. Ozfatura, S. Ulukus, and D. Gündüz, "Straggler-aware Distributed Learningw Communication–computation Latency Trade-off," *Entropy*, vol. 22, no. 5, p. 544, 2020.

[10] X. Su, "Speeding Up Coded Distributed Machine Learning," Ph.D. dissertation, The Graduate Center, City University of New York, 2024.

[11] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 3, pp. 7–11, 2015.

[13] D. Wang, G. Joshi, and G. W. Wornell, "Efficient Straggler Replication in Large-scale Parallel Computing," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 4, no. 2, pp. 1–23, 2019.

[14] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler Mitigation in Distributed Matrix Multiplication: Fundamental Limits and Optimal Coding," *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1920–1933, 2020.

[15] S. Dutta, V. Cadambe, and P. Grover, "'Short-Dot': Computing Large Linear Transforms Distributedly Using Coded Short Dot Products," *IEEE Transactions on Information Theory*, vol. 65, no. 10, pp. 6171–6193, 2019.

[16] X. Su, J. Parker, X. Zhong, X. Fan, and J. Li, "Local re-encoding for coded matrix multiplication," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 1265–1279, 2022.

[17] E. Ozfatura, S. Ulukus, and D. Gündüz, "Coded Distributed Computing with Partial Recovery," *IEEE Transactions on Information Theory*, vol. 68, no. 3, pp. 1945–1959, 2021.

[18] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving Distributed Gradient Descent Using Reed-Solomon Codes," in *Proceedings of IEEE International Symposium on Information Theory (ISIT)*. Institute of Electrical and Electronics Engineers Inc., 2018, pp. 2027–2031.

[19] M. Ye and E. Abbe, "Communication-computation Efficient Gradient Coding," in *Proceedings of 35th International Conference on Machine Learning (ICML)*, 2018, pp. 5610–5619.

[20] E. Ozfatura, D. Gündüz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 2729–2733.

[21] E. Ozfatura, S. Ulukus, and D. Gunduz, "Distributed Gradient Descent with Coded Partial Gradient Computations," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., 2018, pp. 3492–3496.

[22] H. Wang, S. Guo, B. Tang, R. Li, Y. Yang, Z. Qu, and Y. Wang, "Heterogeneity-aware Gradient Coding for Tolerating and Leveraging Stragglers," *IEEE Transactions on Computers*, 2021.

[23] M. N. Krishnan, M. Ebrahimi, and A. Khisti, "Sequential Gradient Coding for Straggler Mitigation," *arXiv preprint arXiv:2211.13802*, 2022.

[24] E. Ozfatura, S. Ulukus, and D. Gündüz, "Distributed gradient descent with coded partial gradient computations," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 3492–3496.

[25] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd," in *International conference on artificial intelligence and statistics*. PMLR, 2018, pp. 803–812.

[26] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," *arXiv preprint arXiv:1604.00981*, 2016.

[27] S. Wang, J. Liu, and N. Shroff, "Fundamental Limits of Approximate Gradient Coding," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–22, 2019.

[28] Z. Charles, D. Papailiopoulos, and J. Ellenberg, "Approximate gradient coding via sparse random graphs," *arXiv preprint arXiv:1711.06771*, 2017.

[29] R. Bitar, M. Wootters, and S. E. Rouayheb, "Stochastic Gradient Coding for Straggler Mitigation in Distributed Learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 277–291, 2020.

[30] X. Su, B. Sukhnandan, and J. Li, "On Arbitrary Ignorance of Stragglers with Gradient Coding," in *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, 2023.

[31] "Anyscale - Ray Distributed Computing - Anyscale." [Online]. Available: https://www.anyscale.com/ray-open-source

[32] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," *arXiv preprint arXiv:1707.08819*, 2017.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[34] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf