Breaking (Global) Barriers in Parallel Stochastic Optimization with Wait-Avoiding Group Averaging

1st Shigang Li  
Department of Computer Science  
ETH Zurich  
Switzerland  
shigang.li@inf.ethz.ch

2nd Tal Ben-Nun  
Department of Computer Science  
ETH Zurich  
Switzerland  
talbn@inf.ethz.ch

3rd Dan Alistarh  
IST Austria  
dan.alistarh@ist.ac.at

4th Salvatore Di Girolamo  
Department of Computer Science  
ETH Zurich  
Switzerland  
salvatore.digirolamo@inf.ethz.ch

5th Nikoli Dryden  
Department of Computer Science  
ETH Zurich  
Switzerland  
ndryden@ethz.ch

6th Torsten Hoefler  
Department of Computer Science  
ETH Zurich  
Switzerland  
torsten.hoefler@inf.ethz.ch

Abstract—Deep learning at scale is dominated by communication time. Distributing samples across nodes usually yields the best performance, but poses scaling challenges due to global information dissemination and load imbalance across uneven sample lengths. State-of-the-art decentralized optimizers mitigate the problem, but require more iterations to achieve the same accuracy as their globally-communicating counterparts. We present Wait-Avoiding Group Model Averaging (WAGMA) SGD, a wait-avoiding stochastic optimizer that reduces global communication via subgroup weight exchange. The key insight is a combination of algorithmic changes to the averaging scheme and the use of a group allreduce operation. We prove the convergence of WAGMA-SGD, and empirically show that it retains convergence rates equivalent to Allreduce-SGD. For evaluation, we train ResNet-50 on ImageNet; Transformer for machine translation; and deep reinforcement learning for navigation at scale. Compared with state-of-the-art decentralized SGD, WAGMA-SGD significantly improves training throughput (by 2.1x on 1,024 GPUs) and achieves the fastest time-to-solution.

Index Terms—stochastic gradient descent, distributed deep learning, decentralized optimization

I. INTRODUCTION

The introduction of deep learning is one of the most important advancements in science over the past two decades, powering industries from autonomous driving [1] to drug discovery [2]. With the rise of deep neural networks, their training evolved into a computationally-intensive task that consumes as many resources as modern complex high-performance computing problems [3]. As a result, an abundance of research has been conducted into its scaling and distribution [4].

The leading contenders for largest workloads in deep learning are Neural Language Models [5], [6], Deep Reinforcement Learning (RL) [7], [8] and Neural Architecture Search [9], [10]. In these regimes, computation time is measured in thousands of “GPU days”, with some utilizing hundreds of accelerators (GPUs, TPUs) for several weeks [11]–[13].

Distributed training is largely supported by data parallelism, where sample evaluation is partitioned across processors. In this mode of parallelism, all participants must exchange their gradients or model, resulting in an Allreduce operation across a cluster [14]. In practice, the exchange communication dominates the overall runtime [12], especially in large minibatch SGD. To exacerbate the problem, certain datasets and environments are inherently imbalanced, e.g., with different sentence/video lengths [15] or heterogeneous environments in RL [16].

In order to mitigate the wait time for gradient/weight exchange, existing approaches attempt to relax model consistency between processors [4], [17]. Examples include synchronous gossip-based SGD [18], [19], asynchronous SGD [20]–[23], and asynchronous SGD with bounded staleness [24]–[27]. Gossip-based SGD replaces the global allreduce by communicating with randomly selected neighbors. Asynchronous SGD breaks the global synchronization to mitigate the effect of stragglers (slow processes). However, most of these approaches adversely impact convergence, necessitating an increase in the number of iterations [19], [28], sometimes to the point where synchronous waits are preferable.

In this paper, we solve this problem by introducing Wait-Avoiding Group Model Averaging (WAGMA) SGD, a novel optimizer that combines group collective communication with bounded staleness, in order to ensure competitive performance with decentralized and asynchronous methods, while retaining the convergence rate of synchronous model-averaging SGD. WAGMA-SGD locally communicates model updates across subgroups of processors, mitigating the need for global communication at every training iteration. Specifically, we propose to use a group allreduce operation for model averaging, in which the fastest process will trigger exchanges within all subgroups. Grouping is performed dynamically to facilitate model update propagation, and as a result not only speeds up communication, but also mitigates the effect of unbalanced workloads, all without harming convergence in practice.

We theoretically prove the convergence of WAGMA-SGD,
showing that, for certain parameter values, its convergence rate is comparable to synchronous SGD with model averaging. Subsequently, we test the algorithm on a supercomputer equipped with GPUs for three different categories of deep learning: supervised image classification on the ImageNet dataset; semi-supervised language modeling on the WMT17 translation dataset; and deep reinforcement learning on the Habitat indoor navigation dataset. We show that both theoretically and empirically, WAGMA-SGD is favorable over other asynchronous algorithms and the baselines, which makes it an excellent approach for scaling up distributed deep learning.

Our main contributions are:

- We propose WAGMA-SGD and realize it based on a wait-avoiding group allreduce operation.
- We theoretically analyze the convergence of WAGMA-SGD.
- Compared with state-of-the-art decentralized SGD, WAGMA-SGD improves the training throughput by up to 2.1x on 1,024 GPUs, and achieves the fastest time-to-convergence for all three evaluated tasks.

II. BACKGROUND AND RELATED WORK

Deep neural networks are primarily trained with mini-batch stochastic gradient descent [29]. Let $b$ be the batch size, $W_t$ the neural network weights at step $t$, $(x_i, y_i)$ a set of samples of size $b$, and $\ell$ a loss function. We compute the loss for each sample as $z_i = \ell(W_t, x_i, y_i)$ and then a stochastic gradient as

$$G_t = \frac{1}{b} \sum_{i=0}^{b} \nabla \ell(W_t, z_i).$$

SGD then iterates steps such that $W_{t+1} = W_t - \eta_t G_t$. In more general terms, first-order stochastic gradient update rules can take different forms (e.g., by adding a momentum term), which is represented as $W_{t+1} = W_t + U(G_t, W_{0, \ldots, t}, t)$. In distributed environments with $P$ processors, $b$ denotes the local batch size per processor. We refer to Ben-Nun & Hoefler [4] for a general overview of distributed deep learning.

Thanks to the robustness of stochastic optimization, in distributed environments one can relax weight updates by varying several axes, trading off communication overhead for convergence. Data-parallel distributed SGD algorithms can be broadly identified by answering the following five questions:

**Q1. What are we averaging?**

There are two typical approaches for aggregating distributed updates: gradient and model averaging. When performing gradient averaging, we simply compute $G_t$ as the average over the global batch size. With standard model averaging, the SGD update is applied locally at the node, and then the resulting model $W_{t+1}$ is averaged over all processors.

Complementary to these approaches is the degree of quantization or sparsity in the exchanged updates. As these concepts are out of the scope of this paper, we refer to Tang et al. [17] for a comprehensive survey.

**Q2. Who is coordinating the averaging?**

Earlier implementations of distributed SGD for deep learning [30] use a centralised coordination architecture, where a parameter server or other coordinator maintains a master copy of the model that workers use. As this approach does not scale to large numbers of processors, a decentralized global clock can be synchronized across workers, where each worker maintains a local replica of the model and communicates updates to other workers directly.

To mitigate the overheads of global communication and synchronization, several decentralized instances of SGD have been proposed, e.g., [18], [19], [22], [28], where each worker maintains a local model but communicates updates in separate schedules, rather than synchronizing globally.

**Q3. How old (stale) can averaged components be?**

In a synchronous system, model or gradient averaging occurs when all processes are on the same training iteration $t$. This does not guarantee that every worker uses the same parameters (i.e., consistent model), however, standard parameter server or globally-coordinated methods ensure all workers have a consistent model. In an asynchronous system, averaging can occur between workers at any point. We thus define the staleness of models/gradients by $\tau$, indicating how many iterations have passed since the produced value’s model was updated. A bounded staleness system mitigates convergence issues with asynchronous systems by ensuring that the difference in the number of training iterations between the slowest and fastest processor is bounded, using $\tau$ as a proxy.

**Q4. How often are we globally averaging?**

While bounded and unbounded staleness SGD variants do not adhere to rigid communication schedules, some algorithms may periodically synchronize all processors’ model replicas. This ensures not only the staleness is bounded by $\tau$ but also the consistency of the model is retained throughout training, mitigating its divergence across processors. In other algorithms, this global consensus is achieved post-training, by choosing the model average or the model with best generalization scores. Note that under this nomenclature, synchronous variants’ global average frequency is one step.

**Q5. How many learners are averaging at every step?**

In the steps between the aforementioned global model averaging period, decentralized SGD variants perform local averages with a certain group (or quorum) size $S$, leveraging the fact that several averaging steps can be performed in parallel.

Removing the global communication bottleneck in decentralized SGD has been shown to enable scaling to tens and even hundreds of nodes [18], [19], [22]. However, performing averaging in pairs does come at the cost of worse convergence: in particular, early proposals on decentralized algorithms [18], [22] lose accuracy with respect to the synchronous baseline at scale, while more recent work [19], [28] observe that the algorithms can achieve full accuracy if executed for more iterations than the synchronous baseline: in particular, they execute between twice and four times more SGD iterations in total, relative to the synchronous baseline, erasing much of the speedup due to increased scalability. This decreased convergence behavior is connected to the analytical bounds provided by these algorithms: while the theoretical conver-
A. Training at Scale

An orthogonal challenge to distributed stochastic optimization is that of unbalanced workloads. Imbalance may be caused by the training system [15], [22], [58] or by the task itself [15], [16]. Training on multi-tenant cloud systems can suffer from performance variability due to resource sharing. Several deep learning tasks, such as video classification and machine translation, have inherent load imbalance, because input/output sequences have different lengths [15]. In deep reinforcement learning, an agent must interact with the environment to generate training data. For RL tasks using heterogeneous environments [16], the runtime of training data generation varies significantly.

Beyond deep learning, allreduces have a long history within the HPC community [59]–[70] and nonblocking versions have been used to improve performance [71]. Particular implementations have become widely-used within the deep learning community, including Baidu-Allreduce [43], NCCL [45], Glou [46], and Horovod [44]. Most deep learning frameworks incorporate support for distributed training, either via parameter servers or allreduces [30], [72]–[74]. Communication compression is another common (and complementary) approach to reducing communication overhead [75]–[84]. Communication may also be impacted by different approaches to partitioning layer parameters, such as model parallelism [85]–[90].

B. Comparison Targets

In Table I we summarize and classify the distributed SGD algorithms most relevant to our work. Algorithms in bold are used for comparison in this work. Since they typically scale and perform better on large-scale systems, we limit our comparison to decentralized algorithms. The algorithms we compare our evaluation with are chosen specifically to be spread across the different answers to the above five questions, prioritizing popular algorithms with proven convergence, both in theory and in practical deep learning applications:

- Allreduce-SGD is the standard data-parallel training.
- Local SGD [26], [27], [54]–[57] performs a fixed number of local iterations of SGD (a hyperparameter determined by the user) and then averages the models over all processes with a standard allreduce. Several variants with different methods for determining the frequency of global averaging exist.
- Decentralized parallel SGD (D-PSGD) [18] uses a ring topology, where each process averages its local model with its two neighbors. Processes advance synchronously with a single global clock.
- Stochastic gradient push (SGP) [19] generalizes the topology used in D-PSGD to support more flexible, asymmetric communication patterns.
- Eager-SGD [15] uses partial collective allreduces over the gradients, allowing at most half processors to contribute stale gradients if not ready.
- Asynchronous decentralized parallel SGD (AD-PSGD) [22] extends the idea of D-PSGD by allowing processors to communicate updates at any point in time. These cover nearly all varieties of consistency and averaging, as well as practical differences in communication patterns.

C. Discussion

Following the discussion on the impact of quorum size on convergence (Q5), it is natural to ask whether performing decentralized averaging in larger groups would be able to provide the best of both worlds, enabling the full convergence of the synchronous algorithm, and the scalability of fully decentralized ones. There are two main barriers to this solution: the first one is at the implementation level, since, to our knowledge, no efficient non-blocking implementation of group model averaging exists. The second is at the application level, since it is not clear whether group averaging would be able

| Coordination | Staleness | Gradient Averaging | Model Averaging |
|-------------|-----------|-------------------|----------------|
| Centralized | None      | Parameter server [31], P3 [32] | —               |
|             | Unbounded | Hogwild! [20], Downpour SGD [30], AASGD [33] | —               |
|             | Bounded   | SSP [24], Rudra [35], Softsync SGD [36], Gaia [37], k-async SGD [38], Qsparse-local-SGD [39], Hybrid sync/async [40] | —               |
| Decentralized, $S = P$ | None | Allreduce-SGD [43]–[46] | BMUF [26] |
|             | Unbounded | — | One-shot SGD [47], SimuParallelSGD [48] |
|             | Bounded | Eager-SGD [15], SFB [49], Gradient lag [50] | — |
| Decentralized, $S = \sqrt{P}$ | None | — | — |
|             | Unbounded | — | — |
|             | Bounded | — | — |
| Decentralized, $S = O(1)$ | None | — | Local SGD [27], [54]–[57] |
|             | Unbounded | GossipGraD [23], Choco-SGD [51] | D-PSGD [18], SGP [19] |
|             | Bounded | CDSDG [53] | AD-PSGD [22], Gossiping SGD [52], SwarmSGD [28] |
to achieve the same convergence as the synchronous solution (both in theory and in practice). In the following sections, we address both of these issues.

III. WAIT-AVOIDING GROUP COMMUNICATION

The allreduce collective operation \cite{14} is defined as a reduction whose results are shared among all participants. Although severval optimizations \cite{43, 60} have been designed to improve the performance of this collective, allreduce poses an implicit global synchronization point, which makes it vulnerable to stragglers during deep learning training. On larger systems, the performance of the compute nodes can be impacted by different internal (e.g., load imbalance) and external factors (e.g., network \cite{91} or OS \cite{92} noise), potentially increasing the synchronization overhead. We define this collective as synchronous allreduce. While non-blocking collectives \cite{93} can alleviate the synchronization overhead, they do not fully remove it and completion still waits. Even if the participating processes are perfectly synchronized, the optimal scaling of an allreduce of size \( N \) is at best \( O(\log P + N) \) for \( P \) processes \cite{12, 94}. Therefore, growing process counts will reduce the parallel efficiency and eventually make the reduction a scaling bottleneck.

A. Wait-Avoiding Group Allreduce

To overcome the synchronization overhead and overall collective cost, we introduce a new class of wait-avoiding group collectives, focusing on group allreduce for the purpose of this work. We relax synchronization by making the collectives externally-triggerable \cite{15, 95}, namely, a collective can be initiated without requiring that all the processes enter it, by externally activating the communication schedule of late processes with activation messages sent by the early ones. Once activated, a group allreduce does not perform a global reduction. Instead, it partially reduces the data within non-overlapping groups of processes, limiting the number of communications needed to implement the group collective.

1) Collective activation: In a wait-avoiding group allreduce, any process can make progress regardless of what the other processes are working on. This wait-avoidance is achieved by the activation component. We call the process reaching the collective call the activator. The activator is in charge of informing all the other processes that an allreduce operation has started and that they have to participate, regardless of whether they reached the collective call-site.

![Fig. 1: Wait-avoiding group allreduce on four processes with a group size of two. P1 arrived first and activates the operation.](image)

In a wait-avoiding group allreduce, any process can initiate the collective. We use a modified version of the recursive doubling algorithm that builds a butterfly topology, which can be seen as a set of overlapping binomial trees, one rooted at each process. Any node can activate the collective by sending activation messages along the binomial tree rooted at itself. Fig. 1 shows an example where P1 is the activator. In this case, P1 uses its broadcast tree and sends the activation messages to P0 and P3. Once activated, P0 first forwards the activation message to P2, after which it starts executing its group allreduce schedule.

It is possible that several processes arrive at the wait-avoiding group allreduce operation at close proximity, which means we may have more than one activator during the activation phase. To guarantee that a process does not execute the same collective twice, we assign each operation a version number that is increased every time the collective is executed. The collective version number is encoded in the tag of the activation messages: once an activation is received, the collective schedule is activated only if its version number is lower or equal than the one carried by the activation message.

The version number check is executed also when a process reaches the collective call: if it fails, then the version of the collective that the process wants to activate has already been executed (and the process has passively participated in it). In this case, no activation messages are sent.

2) Asynchronous execution: To enable asynchronous execution of the custom collectives, we extend the fflib communication library \cite{95}, adding support for wait-avoiding group allreduce. fflib allows programmers to customize collective operations via a flexible, DAG-based representation of point-to-point and local compute operations, defined as schedules. The library provides a C-based interface for schedule creation and nonblocking invocation, using MPI as its primary backend, with additional support for network offloading engines such as sPIN \cite{96}. Our defined schedule for group operations models both the activation and group allreduce phases.

B. Dynamic Grouping Strategy

As discussed in Section II, in data-parallel SGD variants such as allreduce SGD \cite{44, 97} and gossip SGD \cite{18, 19, 22}, each process keeps propagating local model updates to all the other processes at every iteration to make global progress. We propose a dynamic grouping strategy to reduce the latency (in steps) of local update propagation. Together with the group allreduce operation, the grouping strategy guarantees that the local updates can be globally propagated within \( \log P \) iterations. The larger the group size, the faster the updates are propagated. By carefully selecting the group size, we can achieve both lower latency than gossip SGD and efficient communication by reducing contention.

We define the dynamic grouping strategy in Algorithm 1. We assume the number of processes \( P \) is a power-of-two, which is a common case in current distributed training systems. The group size \( S (\leq P) \) is also set to a power-of-two. In line 2, we initialize the mask, and calculate the number of phases in a butterfly topology for \( P \) and \( S \) processes, respectively. Line 3 initializes the shift. In each training itera-
Thus, weights. In the following section, we present the algorithm in detail and further discuss this periodic synchronization.

IV. WAIT-AVOIDING GROUP MODEL AVERAGING SGD

Based on the insight that larger groups converge faster, and on the novel implementation of wait-avoiding group collectives, we design the Wait-Avoiding Group Model Averaging (WAGMA-SGD) algorithm. The algorithm can be classified as a model-averaging, bounded-staleness, decentralized SGD with a group size of $S \propto \sqrt[τ]{P}$ and a global communication period of $τ$ steps. As listed in Algorithm 2, WAGMA-SGD is similar to minibatch SGD, but makes a few important distinctions.

In lines 3–7, each process calculates the local gradients $G_{i}^{local}$ and then applies the local gradients to derive and apply the model update $ΔW_i$, as in distributed SGD. Subsequently, the wait-avoiding group model averaging is conducted (lines 8–17) using the aforementioned wait-avoiding communication scheme. From an algorithmic perspective, WAGMA-SGD does not rely on certain choice of group members for the local
collectives. However, instead of randomly choosing groups of processes, we use the butterfly strategy (Algorithm 1) for topology-aware, efficient, deterministic communication.

In each iteration, faster processes will trigger the model averaging immediately without waiting (line 9, $t$ is used to control grouping), which may incur averaging the local models with some stale models from the slower processes. To both bound the staleness and mitigate divergence across local model replicas, we define a synchronization period $\tau$, in which the models are averaged across all the processes using a global allreduce (line 16). Empirically, we set the synchronization period $\tau$ to 10 training iterations, which balances model accuracy with training throughput, as we will show in Section V.

An execution snapshot of WAGMA-SGD ($P = 4$ and $S = 2$) is presented in Fig. 3. Suppose P1 is a straggler. When the group allreduce in iteration $t$ is triggered by any of the other three processes, P1 can only contribute the stale model parameters $W_{t-1}'$. In iteration $t$, P1 and P0 are in the same group; therefore, $W_{t-1}'$ and $W_0''$ will be added together to derive $W_{t \text{sum}}$. P0 will use the averaged model $W_{t+1}^0$ for the next iteration of training. P1 subsequently finishes the calculation for the local updated model in iteration $t$ (i.e., $W_t'$), but finds out that the group allreduce in iteration $t$ is already finished. In this case, it will average the stale model $W_t'$ with $W_{t \text{sum}}$ (line 13 in Algorithm 1), and the averaged model $W_{t+1}'$ will be used for the next iteration of training. Meanwhile, the data in the send buffer of P1 is updated by $W_t'$. If the group allreduce in iteration $t+1$ is triggered by some faster process at this time, P1 will continue to passively contribute the stale model $W_t'$. When a standard allreduce is called at the synchronization point, it forces all the processes to contribute the model parameters after training for the same number of iterations. In Fig. 3, P1 catches up with the other processes in iteration $t+1$; thus, it will contribute the timely model $W_{t+1}'$ to P3, as they are in the same group.

A. Proof of Convergence

a) Algorithm Modelling: For analysis purposes, we will model the algorithm’s execution as follows. We will proceed in steps, indexed by time $t \geq 0$. Each node $i$ maintains its own local model $W_i^t$, and has a local partition of the data. In each step, a group of nodes of size $S$ is chosen to interact. Each node takes a local gradient step, and then nodes average their models. This averaging step might be inconsistent, as per the above semantics.

In the analysis, we will assume that the group of $S$ interacting nodes is chosen uniformly at random—in the long run, the resulting interaction graph will have the same properties as the butterfly interaction strategy used in the implementation. While our analysis considers each interaction sequentially, in practice $O(P/S)$ interaction steps can occur in parallel.

b) Setup and Analytic Assumptions: We will assume a standard setting in which we are given a dataset of $D$ samples $D = \{e_1, e_2, \ldots, e_D\}$, and to each associate a differentiable loss function $f_e : \mathbb{R}^d \rightarrow \mathbb{R}$. Each node $i$ is given a random partition $D_i$ of the dataset $E$, and we wish to solve an empirical risk minimization problem by finding

$$x^* = \arg\min_{x \in \mathbb{R}^d} \left[ f(x) := \frac{1}{D} \sum_{e \in D} f_e(x) \right].$$

Let $F_i = \frac{P}{D} \sum_{e \in D_i} f_e$ be the loss function corresponding to the dataset of the $i$th node, and $F^* = F(x^*)$. To make the analysis tractable, we will make the following standard assumptions on the loss function.

Assumption 1. We assume the following hold:
1) (Lipschitz gradients) All functions $f_e$ have $L$-Lipschitz gradient, for some constant $L$.
2) (Bounded Second Moment) There exists a constant $M$ such that for any node $i$ and $w \in \mathbb{R}^d$, $E_{e \in D_i} \| \nabla f_e(w) \|^2 \leq M^2$.
3) (Bounded Staleness) The staleness during the averaging step is upper bounded by a parameter $\tau$. That is, for any node $i$ participating in the averaging step at time $t$, averaging is performed with respect to model $W_i^{t'}$, where $t' \geq t - \tau + 1$, and every gradient update is applied globally at most $\tau$ steps after it was generated.

c) Convergence result: We can now state the main convergence result. For readability, we state a simplified variant that highlights the relationship between parameter values, in particular the relation between the convergence time $T$, $P$ processors, and the size of the interacting group $S$. The full statement and its proof will be integrated in the paper.

Theorem 1. Consider the setting described above, in which we wish to optimize a non-convex function $F : \mathbb{R}^d \rightarrow \mathbb{R}$. Let $S$ be the size of a communication group, and assume that the maximum staleness $\tau$ is constant. Fix a success parameter $\varepsilon > 0$. For each time $t$, we define $\mu_t = \sum_{i=1}^P W_i^t$ to be the average of local models at time $t$. Then there exists a setting of the learning rate such that, if the algorithm has taken
\[ T = \Omega \left( \max \left\{ \frac{P^6 L^2}{S^2 M^2}, \frac{(F(W_0) - F^*)M^2}{\epsilon^2} \right\} \right) \text{ steps,} \]

then there exists an iterate \( 0 \leq T^* \leq T \) such that

\[ \mathbb{E}[\|\nabla F(\mu_{T^*})\|^2] \leq \epsilon, \]

where the expectation is taken w.r.t. the randomness in the sampling and interactions.

d) Discussion: At a high level, this claim shows that the algorithm will eventually reach a point where the model average has negligible gradient, i.e., is at a local minimum. While this does not guarantee convergence to a global minimum, it matches the best achievable guarantees for SGD in the non-convex setting [98]. The convergence proof follows the general decentralized asynchronous framework of Lian et al. [22], with differences due to the specific structure of the group communication pattern we employ, and the asynchronous nature of wait-avoiding collectives. Further, we note that the convergence guarantee can be extended to 1) apply to individual models instead of the model average; and 2) relax the bounded second moment assumption to a bound on the variance. Both these improvements come at the cost of additional technical complexity, so we will defer them in the full version of our convergence proof. The current statement of the theorem obscures the rate at which convergence occurs — for standard parameter settings, the convergence speedup (i.e., the rate at which we get to a point of negligible gradient) with respect to the number of nodes is linear. This linear speedup is the best possible, and matches the rates for other decentralized algorithms, e.g. [18], [22]. We refer the interested reader to the full text of our convergence proof.

It is interesting to examine the impact of the group size \( S \) on convergence: in particular, notice that the time to convergence decreases quadratically in \( S \). Specifically, assuming that group size is small, say, \( S = 2 \), and other parameters are constant, then the number of steps to reach a local optimum is \( \Omega(P^6) \). This matches the best known bounds in the decentralized model for pairwise interactions [18], [22]. However, \( \Omega(P^6) \) can exceed the number of SGD steps taken during regular training even for moderate node counts, making the bound meaningless. For example, the number of SGD steps when training ResNet50 on ImageNet is around 5 \( \cdot \) 10^6, making the bound meaningful only for \( P < 10 \). For larger group size, e.g., \( S \) in the order of \( P \), our analysis decreases this step bound to \( \Theta(P^4) \), which asymptotically matches the convergence rate and step bound when model averaging is performed synchronously and globally (e.g., the bound of [18] for all-to-all communication), and is also practically more relevant.

V. EXPERIMENTAL EVALUATION

We conduct our experiments on the CSCS Piz Daint supercomputer. Each Cray XC50 compute node contains a 12-core Intel Xeon E5-2690 CPU with 64 GB RAM, and one NVIDIA Tesla P100 with 16 GB memory. The compute nodes are connected by Cray Aries interconnect in a Dragonfly topology. The communication library is Cray MPICH 7.7.2. We use one MPI process per node and utilize the GPU for acceleration in all following experiments. We evaluate three different deep learning problems, including image classification (ResNet-50 [99] on ImageNet [100]), machine translation (Transformer [101] on WMT17), and deep reinforcement learning (PPO [16], [102] for navigation in Habitat [103]). For throughput tests, we run the number of nodes until reaching a point where batch size is too large to converge [104].

A. Baselines

We compare our WAGMA-SGD with the state-of-the-art data-parallel SGD variants, including Allreduce-SGD [44], [97], local SGD [26], [54], gossip-based SGD variants (D-PSGD [18], AD-PSGD [22], and SGP [19]), and eager-SGD [15]. Unless mentioned specifically, the synchronization period of local SGD is set to one, namely calling a standard allreduce to average the models in each training iteration, which essentially is a synchronous SGD. For SGP, we evaluate its performance with different number of communication neighbors [19]. For more detailed discussion about the baselines, please refer to Section II.

B. Image Classification with Simulated Workload Imbalance

Residual Networks (ResNet) [99] are pervasively used in computer vision tasks. To evaluate their performance, we train ResNet-50 on ImageNet (total 25,559,081 trainable parameters) with a local batch size of 128 images. Although the training workload is balanced due to the input size being fixed, performance variability is observed in practice when training on multi-tenant cloud systems [15], [22], [105]. To simulate the same degree of imbalance, we randomly select two processes at every training step to inject a certain amount of delay (320 ms), according to the performance variability on cloud machines [15]. For WAGMA-SGD, we set the synchronization period \( \tau = 10 \), and the group size \( S = \sqrt{P} \). Both \( P \) and \( S \) are power-of-two in our experimental configuration.

![Fig. 4: Throughput comparison between different parallel SGD algorithms for ResNet-50 on ImageNet with simulated load imbalance. Local batch size is 128.](image)

Fig. 4 shows the training throughput as the number of GPU nodes increases from 4 to 256, and the top of the
rectangular wrapping each cluster indicates the ideal throughput with minimal communication overhead. Compared with local SGD, Allreduce-SGD (implemented in Deep500 [97]), D-PSGD, SGP (two communication neighbors), and eager-SGD when training on 64 GPU nodes, WAGMA-SGD achieves 1.25x, 1.26x, 1.23x, 1.25x, and 1.13x speedup, respectively. The speedup becomes larger as the number of GPU nodes increases to 256: WAGMA-SGD achieves up to 1.37x speedup. The only algorithm with higher throughput than WAGMA-SGD is AD-PSGD, in which the asynchronous communication is completely overlapped with the computation. These results show that WAGMA-SGD can better handle the unbalanced workload than the synchronous SGD algorithms (i.e., local SGD, Allreduce-SGD, D-PSGD, and SGP), as well as the bounded-staleness eager-SGD variant. In the latter case, while staleness is bounded, the algorithm still conducts a global collective communication for gradient averaging in each training iteration. In contrast, WAGMA-SGD keeps the collectives within each group, and thus has a better parallel scalability.

By setting the group size \( S = \sqrt{P} = 8 \), WAGMA-SGD has a faster model update propagation speed (globally propagate only using \( \log S P = 2 \) iterations) than the gossip-based algorithms (globally propagate using at least \( \log S P = 6 \) iterations), which makes WAGMA-SGD achieve higher accuracy. This is consistent with our analysis in Section IV-A.

To further analyze the convergence properties of WAGMA-SGD, we conduct additional experiments. 1. In the first experiment, we remove the wait-avoiding group collectives in WAGMA-SGD and only keep standard allreduce operations on the synchronization points, which is essentially equivalent to local SGD with a synchronization period \( \tau = 10 \). This causes the top-1 validation accuracy to sharply drop to 66.9%. 2. In a second experiment, we execute group model averaging without using the dynamic grouping strategy (i.e., the groups are fixed). In this case, the top-1 validation accuracy drops to 73.5%. 3. We also experiment with increasing the group size to 64 (i.e., a global collective). While accuracy does not increase, the throughput drops by factor of 1.07x. 4. Lastly, we decrease the group size to 4 and observe that the top-1 validation accuracy drops to 72.8%.

The results from experiments 1 and 2 indicate that the combination of group allreduce operations and the dynamic grouping strategy is essential to achieve good generalization performance. The results from experiments 3 and 4 demonstrate that \( S = \sqrt{P} \) empirically exhibits the best performance among different group size settings.

C. Machine Translation

Transformers are sequence-to-sequence transducers that can be used to translate a sequence of words from one language to another. We use the standard-sized Transformer network [101], which has 61,362,176 trainable parameters, to train English to German translation WMT17 dataset. While training the model, the computation overhead changes with the length of the input and output sentences. The samples in the training dataset typically consist of sentences in various lengths and thus the training workload is unbalanced. As shown in Fig. 6, even when using a bucketing strategy to group sentences with similar lengths, there is a high variance in workload size between samples. Specifically, in our experiment each local batch contains equal number of sentences sampled from a randomly selected bucket, where the maximum local batch size is set to 8,192 tokens. For WAGMA-SGD, we set the synchronization period \( \tau = 8 \) and the group size \( S = \sqrt{P} \).

Fig. 7 presents the training throughput as the number of GPU nodes increases from 4 to 64, where the top of the rectangle indicates the ideal throughput without communication overhead. On 16 GPU nodes, WAGMA-SGD achieves the highest throughput, compared with local SGD, Allreduce-SGD (implemented in Horovod [44]), D-PSGD, AD-PSGD, and SGP (one communication neighbor). When the number of GPU nodes increases to 64, as with image classification WAGMA-SGD exhibits a lower throughput than AD-PSGD but higher than all the other variants. Observe that on 64 nodes, all the algorithms perform far worse than the ideal

Fig. 5 presents the Top-1 validation accuracy (in accordance with MLPerf [106]) when training for 90 epochs on 64 nodes with a total batch size of 8,192. We can see that the accuracy of WAGMA-SGD (75.0%) is very close to the standard Allreduce-SGD (75.8%) and local SGD (75.3%), but WAGMA-SGD significantly reduces the training time. Gossip-based SGD algorithms, such as D-PSGD and the higher-throughput AD-PSGD, attain much lower accuracy than the other variants. This can be explained by the fact that the algorithms have not fully converged, requiring more steps to be taken to achieve comparable accuracy [28]. For SGP, we set and tune the number of communication neighbors to achieve the highest generalization using a directed exponential graph [19], which causes it to achieve higher accuracy than D-PSGD and AD-PSGD, yet still lower than WAGMA-SGD. Note that the default setting for the number of communication neighbors in SGP is one, whereas we set it to two for better generalization performance. Overall, WAGMA-SGD achieves the highest accuracy-vs-time among all parallel SGD variants.

![Fig. 5: Top-1 validation accuracy of ResNet-50 on ImageNet training for 90 epochs using 64 GPU nodes. Each point is at the boundary of every 10 epochs.](image-url)
throughput. We believe that this effect stems from the balance of the number of parameters (occupying 245 MB alone) vs. the operational intensity to compute backpropagation. Since transformer networks mostly consist of tensor contractions implemented as batched matrix products, which utilize GPUs well, communication overhead dominates and not even AD-PSGD manages to overlap communication with computation.

As for accuracy, Fig. 8 presents the BiLingual Evaluation Understudy (BLEU) score (higher is better) on the test dataset after training for 10 epochs on 16 nodes. As the total batch size is a relatively large number of tokens (131,072), it incurs similar quality degradation as in other deep learning problems [104]. Still, among all SGD variants, WAGMA-SGD achieves the highest score using the shortest training time. Gossip-based SGD variants, including D-PSGD, AD-PSGD, and SGP (1n, i.e., one communication neighbor), have lower score than the others, likely because of the slower model update propagation. To verify this claim, we increase the number of communication neighbors to two in SGP (2n), which improves the score to 24.5 (equivalent to local SGD). However, this accuracy increase comes at the cost of significantly reduced training speed compared with SGP (1n).

We conduct additional experiments for WAGMA-SGD, similarly to Section V-B: (1) Without using the dynamic grouping strategy (i.e., fixed groups), the score drops to 23.8; (2) By increasing the group size to 16 (i.e., global collective), accuracy does not improve and training throughput drops by a factor of 1.28x; and (3) By decreasing the group size to 2, the score drops to 23.3. These results reaffirm the conclusions from image classification.

D. Deep Reinforcement Learning

Due to the inherent characteristics of the problem, reinforcement learning poses a more challenging training process over supervised and semi-supervised learning. This also applies to the heterogeneity in workloads during training — since the problems in question involve interacting with an environment in episodes (where failure terminates an episode early), a variety of episode lengths may occur within a single minibatch, in a way that cannot be anticipated or categorized in buckets.

We use the popular Proximal Policy Optimization (PPO) policy gradient optimizer [102] to train a model for robot navigation on a meta-dataset called Habitat [103], which is composed of multiple heterogeneous environments. We first confirm previous claims [16] and our own in Fig. 9, where we collate the runtime distribution of 5,000 training iterations. The runtime is very widely distributed: from 1.7 seconds to 43.5, with a median below 2 seconds, which makes it an excellent use case for the load-rebalancing properties of WAGMA-SGD.

To evaluate the performance, we train a standard ResNet-LSTM model for navigation. In particular, the network structure is composed of a ResNet-18 visual encoder, connected
to a stack of two Long Short-Term Memory (LSTM) [107] recurrent units functioning as the policy, containing 8,476,421 trainable parameters. The measured heterogeneous environments in Habitat, Gibson [108] and Matterport3D [109], consist of interactive RGB-D datasets. We set the experience steps to 128 and use the two vectorized (namely, optimized) environments, which means each GPU node needs to collect 256 experience steps for each training iteration. We set the WAGMA-SGD synchronization period to $\tau = 8$.

Fig. 10 presents the training throughput as the number of GPU nodes increases from 16 to 1024, where the top of the rectangle indicates the ideal throughput without communication overhead. Compared with local SGD, D-PSGD, and SGP (four communication neighbors) on 1,024 GPU nodes, WAGMA-SGD achieves 2.33x, 1.88x, and 2.10x speedup, respectively. The violin plot shows the throughput distribution. WAGMA-SGD only has lower throughput than AD-PSGD, since AD-PSGD is fully asynchronous. These results show that WAGMA-SGD excels in handling highly unbalanced workloads, achieving good scaling efficiency.

Complementary to performance, we study the Success weighted by Path Length (SPL) score (higher is better) after training the model for 10 hours on 64 GPUs. All models are tested four separate times to account for variability, and the average scores together with the standard deviation (shaded regions) are plotted in Fig. 11. As the figure shows, despite the scalability of AD-PSGD, it only achieves 0.051 SPL on average, and seems to converge, deeming it unusable for RL problems. On the other hand, WAGMA-SGD achieves the highest score over time, even over local SGD. A possible reason for this might be asynchronous methods tend to overcome local convergence issues in deep reinforcement learning [21]. This is also seen in SGP, which scores higher than local SGD, but not as well as WAGMA-SGD, whose quorum size is larger.

Beyond our experiments, the current state-of-the-art SPL score is 0.922 [16]. This score is achieved after training on 2.5 billion experience steps. WAGMA-SGD consumes total 2.6 million experience steps after training for 10 hours using 64 GPUs, and achieves on average 83.1% (up to 91.2%) of the SotA score, and the results seem to keep increasing.

This indicates that we are able to achieve almost equivalent generalization in 3 orders of magnitude fewer iterations.

VI. COLLECTIVES IN CONTEXT

Collective operations have a core role in running applications efficiently at scale. As such, their optimization has led to several implementation and algorithmic variants.

Blocking collectives [14] constitute the basic class of operations. In this case, the collective call is allowed to return only when the calling process has completed all the actions needed to its participation in the operation. A first optimization to blocking collectives is to make them non-blocking [93], enabling processes to return immediately and overlap other activities with the ongoing collective.

Some collectives require all the processes to invoke it in order to complete, e.g., a reduction cannot be computed before knowing all the values to reduce. Hence, their completion time can be influenced by any skewing (imbalance) among the processes. Solo collectives [95] remove this synchronization overhead by making the collectives externally-triggerable: once a process joins the collective, it sends an activation message to all the others, making them to start the collective independently from their state. An issue of solo collectives is that they make triggering the collective possible, even if there is only one process joining it. Majority collectives [15] extend the solo idea by requiring that at least $P/2$ processes join the collective before triggering it. While these collectives are not guaranteed to be equivalent to their blocking or non-blocking counterparts, they are suited for machine learning tasks, due to the robustness of stochastic optimization to staleness.

Both solo and majority collectives aim to minimize the synchronization overhead. However, once activated, the collective is fully performed, making the application pay the full operation cost plus the activation overhead. Wait-avoiding group collectives (this work) utilize the approach of solo collectives to achieve asynchrony, and further reduce the overall operation cost by dynamically selecting subgroups of processes, each of which executing the collective independently from the others.
We show, both theoretically and in practice, that stochastic optimization via group model averaging — averaging the learned weights across subgroups of nodes — functions well in large clusters. We prove that the algorithm converges under the standard conditions of SGD, and through a careful implementation of wait-avoiding collectives, we use the topology of the network to attain the best scaling results without losing accuracy. For the same number of steps, WAGMA-SGD achieves equivalent (or higher) generalization scores as synchronous SGD, while still achieving up to 2.1× speedup (on RL) over the previous state-of-the-art, gossip-based SGD. Similar results are observed in other models from various subfields, empirically proving that this approach is the first to successfully tackle deep learning in extreme scales, dispensing with step-wise global synchronization and bringing SGD to the regime of supercomputers.

REFERENCES

[1] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and Z. Zieba, “End to end learning for self-driving cars,” 2016.

[2] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. T. Jones, T. Schaul, K. Kavukcuoglu, and D. Hassabis, “Improved protein structure prediction using potentials from deep learning,” Nature, vol. 577, no. 7792, pp. 706–710, 2020.

[3] D. Amodei and D. Hernandez, “AI and compute,” May 2018, https://openai.com/blog/ai-and-compute/.

[4] T. Ben-Nun and T. Hoefler, “Densifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis,” ACM Comput. Surv., vol. 52, no. 4, Aug. 2019. [Online]. Available: https://doi.org/10.1145/3320060

[5] J. Kaplan, S. McCandlish, T. Hnighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, “Scaling laws for neural language models,” 2020.

[6] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, “Communication-efficient distributed deep learning: A comprehensive survey,” 2020.

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in International Conference on Machine Learning, 2016, pp. 1928–1937.

[8] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” in Proceedings of the 35th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmedias, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3034–3032. [Online]. Available: http://proceedings.mlr.press/v80/lian18a.html

[9] S. Li, T. Ben-Nun, S. D. Girolamo, D. Alistarh, and T. Hoefler, “Taming unbalanced training workloads in deep learning with partial collective operations,” in Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2020.

[10] H. Liu, K. Simonyan, and Y. Yang, “Darts: Differentiable architecture optimization using potentials from deep learning,” in Advances in Neural Information Processing Systems 24, 2011, pp. 693–701.

[11] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, “Neural architecture search,” in The Thirty-Third AAAI Conference on Artificial Intelligence, (AAAI 2019). AAAI Press, 2019, pp. 4780–4789.

[12] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, “Communication-efficient distributed deep learning,” in Advances in Neural Information Processing Systems 24, 2011, pp. 693–701.

[13] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, “Communication-efficient distributed deep learning: A comprehensive survey,” 2020.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in International Conference on Machine Learning, 2016, pp. 1928–1937.

[15] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” in Proceedings of the 35th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmedias, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3034–3032. [Online]. Available: http://proceedings.mlr.press/v80/lian18a.html
A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep networks," in Advances in neural information processing systems (NeurIPS), 2012, pp. 1223–1231.

M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scalable distributed learning with the parameter server;," in Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 583–598. [Online]. Available: http://dl.acm.org/citation.cfm?id=2685048.2685095

A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko, "Priority-based parameter propagation for distributed DNN training," in Proceedings of the 2nd SysML Conference, 2019.

D. Grischchenko, F. Iutzeler, J. Malick, and M.-R. Amini, "Asynchronous distributed learning with sparse communications and identifiation," arXiv preprint arXiv:1812.03871, 2018.

Z. Tang, S. Shi, and X. Chu, “Communication-efficient decentralized learning with sparsification and adaptive peer selection,” arXiv preprint arXiv:2002.09692, 2020.

S. Gupta, W. Zhang, and F. Wang, “Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in 2016 IEEE 16th International Conference on Data Mining (ICDM), 2016.

W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-SGD for distributed deep learning," in 25th International Joint Conference on Artificial Intelligence (IJCAI), 2016.

K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning: Approaching LAN speeds," in Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI'17. Berkeley, CA, USA: USENIX Association, 2017, pp. 629–647. [Online]. Available: http://dl.acm.org/citation.cfm?id=3154630.3154682

S. Dutta, G. Joshi, S. Ghosh, P. Duhe, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD," in Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), 2018.

D. Basu, D. Data, C. Karakus, and S. Diggavi, “Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations;" in Advances in Neural Information Processing Systems (NeurIPS), 2019.

T. Kurth, J. Zhang, N. Satish, E. Racah, I. Mitilagkas, M. M. A. Patwary, T. Malas, N. Sundaram, W. Bihimi, M. Smorkolov et al., “Deep learning at 15PF: supervised and semi-supervised classification for scientific data;" in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2017.

H. B. McMahan, E. Moore, D. Ramage, S. Hampson et al., “Communication-efficient learning of deep networks from decentralized data;" in Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), 2017.

J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency;" in NeurIPS Workshop on Private Multi-Party Machine Learning, 2016.

A. Gibiansky, “Bringing hpc techniques to deep learning;" URL http://research.baidu.com/combrging-hpc-techniques-deep-learning, 2017.

A. Sergeev and M. Del Balso, “Horovod: fast and easy distributed deep networks in TensorFlow;" arXiv preprint arXiv:1810.05799, 2018.

NVIDIA, “NVIDIA collective communications library;" 2020. [Online]. Available: https://developer.nvidia.com/nvcc

Facebook, “Gloo;" 2018. [Online]. Available: https://github.com/facebookincubator/gloo

R. Mcdonald, M. Mohri, N. Silberman, D. Walker, and G. S. Mann, “Efficient large-scale distributed training of conditional maximum entropy models;" in Advances in neural information processing systems (NeurIPS), 2009.

M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, “Parallelized stochastic gradient descent;" in Advances in neural information processing systems (NeurIPS), 2010.

P. Xie, J. K. Kim, Y. Zhou, Q. Ho, A. Kumar, Y. Yu, and E. Xing, “Lighter-communication distributed machine learning via sufficient factor broadcasting;" in Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, ser. UAI’16. Arlington, Virginia, United States: AUAI Press, 2016, pp. 795–804. [Online]. Available: http://dl.acm.org/citation.cfm?id=3020948.3021030

T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica et al., “Exascale deep learning for climate analytics," in International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2018.

A. Koloskova, S. U. Stich, and M. Jaggi, “Decentralized stochastic optimization and gossip algorithms with compressed communication;" arXiv preprint arXiv:1902.02040, 2019.

P. H. Jin, Q. Yuan, F. N. Iandola, and K. Keutzer, “How to scale distributed deep learning?" in Workshop on Machine Learning Systems at NeurIPS 2016, 2016.

Z. Jiang, A. Balu, C. Hegde, and S. Sarkar, “Collaborative deep learning in fixed topology networks;" in Advances in Neural Information Processing Systems (NeurIPS), 2017.

T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi, “Don’t use large mini-batches, use local sgd;" arXiv preprint arXiv:1808.07217, 2018.

F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. Cadambe, “Local SGD with periodic averaging: Tighter analysis and adaptive synchronization;" in Advances in Neural Information Processing Systems (NeurIPS), 2019.

J. Wang and G. Joshi, “Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD;" in Proceedings of the Second SysML Conference, 2019.

A. Reisizadeh, H. Taheri, A. Mokhtari, H. Hassani, and R. Pedarsani, “Robust and communication-efficient collaborative learning;" in Advances in Neural Information Processing Systems (NeurIPS), 2019.

A. Iosup, N. Yigitbasi, and D. Epema, “On the performance variability of production cloud services;" in 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE, 2011, pp. 104–113.

R. Thakur, R. Rabenseifner, and W. Gropp, “Optimization of collective communication operations in mpich;" The International Journal of High Performance Computing Applications, vol. 19, no. 1, pp. 49–66, 2005.

R. Rabenseifner, “Optimization of collective reduction operations;" in International Conference on Computational Science. Springer, 2004, pp. 1–9.

M. Barnett, L. Shaler, R. van De Geijn, S. Gupta, D. G. Payne, and J. Watts, “Interprocessor collective communication library (interCom);" in Proceedings of IEEE Scalable High Performance Computing Conference. IEEE, 1994.

M. Shoff and R. A. Van De Geijn, “CollMark: MPI collective communication benchmark;" in International Conference on Supercomputing, 2000.

V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir, “CCL: A portable and tunable collective communication library for scalable parallel computers;" IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 2, 1995.

E. Chan, R. van de Geijn, W. Gropp, and R. Thakur, “Collective communication on architectures that support simultaneous communication over multiple links;" in PPOPP. ACM, 2006.

E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn, “Collective communication: theory, practice, and experience;" Concurrency and Computation: Practice and Experience, vol. 19, no. 13, 2007.

N. Jain and Y. Sabharwal, “Optimal bucket algorithms for large MPI collectives on torus interconnects;" in ICS. ACM, 2010.

C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, “Image classification at supercomputer scale;" in NeurIPS Systems for ML Workshop, 2018.

A. A. Awan, C.-H. Chu, H. Subramoni, and D. K. Panda, “Optimized broadcast for deep learning workloads on dense-GPU InfiniBand clusters;" in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2017, pp. 1–11.

T. Hoefler, J. Squyres, W. Rehm, and A. Lumsdaine, “A Case for Non-Blocking Collective Operations;" in Frontiers of High Performance
D data in indoor environments,” *International Conference on 3D Vision (3DV)*, 2017.