Bagua: Scaling up Distributed Learning with System Relaxations

Shaoduo Gan*, Jiawei Jiang†, Binhang Yuan, Ce Zhang
ETH Zürich, Switzerland
{firstname.lastname}@inf.ethz.ch

Xiangru Lian*, Rui Wang, Jianbin Chang, Chengjun Liu, Hongmei Shi, Shengzhuo Zhang, Xianghong Li, Tengxu Sun, Sen Yang, Ji Liu
kuaishou technology, China
admin@mail.xrlian.com;ji.liu.uwisc@gmail.com

ABSTRACT

Recent years have witnessed a growing list of systems for distributed data-parallel training. Existing systems largely fit into two paradigms, i.e., parameter server and MPI-style collective operations. On the algorithmic side, researchers have proposed a wide range of techniques to lower the communication via "system relaxations": *quantization*, *decentralization*, and *communication delay*. However, most, if not all, existing systems only rely on standard synchronous and asynchronous stochastic gradient (SG) based optimization, therefore, cannot take advantage of all possible optimizations that the machine learning community has been developing recently. Given this emerging gap between the current landscapes of systems and theory, we build Bagua, a MPI-style communication library, providing a collection of primitives, that is both flexible and modular to support state-of-the-art system relaxation techniques of distributed training. Powered by this design, Bagua has a great ability to implement and extend various state-of-the-art distributed learning algorithms. In a production cluster with up to 16 machines (128 GPUs), Bagua can outperform PyTorch-DDP, Horovod and BytePS in the end-to-end training time by a significant margin (up to 2×) across a diverse range of tasks. Moreover, we conduct a rigorous tradeoff exploration showing that different algorithms and system relaxations achieve the best performance over different network conditions.

PVLDB Reference Format:
Shaoduo Gan, Xiangru Lian, Rui Wang, Jianbin Chang, Chengjun Liu, Hongmei Shi, Shengzhuo Zhang, Xianghong Li, Tengxu Sun, Jiawei Jiang, Binhang Yuan, Sen Yang, Ji Liu, Ce Zhang. Bagua: Scaling up Distributed Learning with System Relaxations. PVLDB, 15(4): 804 - 813, 2022.
doi:10.14778/3503585.3503590

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/BaguaSys/bagua.

1 INTRODUCTION

The increasing performance of distributed machine learning systems has been one of the main driving forces behind the rapid advancement of machine learning techniques. From AlexNet [35] in 2012 to GPT-3 [11] in 2020, each leap in model quality is enabled by the growth of both the model size and the amount of data one can train a model with, along with a rapid increase in computations [47]. Behind this improvement are two major enabling factors: hardware accelerations (e.g., GPUs and TPUs) and the development of efficient and scalable distributed training algorithms [4, 7, 72, 73, 76] It is not unfair to say that a scalable distributed training system is the cornerstone of modern deep learning techniques.

In this paper, we scope ourselves and focus on *data parallel training*, one of the most popular distributed training paradigms in which the data set is partitioned across different workers and the model fits into a single device. Not surprisingly, recently years have witnessed a growing list of systems for distributed data parallel training. Existing systems fit into two paradigms, following the seminal work done by Li et al. [38] on *parameter server* and Sergeev et al. [56] on using MPI collective operations such as *Allreduce*. Both paradigms have enabled industrial-scale distributed training systems [47]: Adam (Microsoft) [13], early TensorFlow (Google) [3], Poseidon (Petuum) [77], Angel (Tencent) [32], and BytePS (ByteDance) [33] are based on *parameter server*, while PyTorch-DDP (Facebook) [39], Mariana (Tencent) [82], MALT (NEC Labs) [37], NCCL (NVIDIA) [2], and Horovod (Uber) [56] are based on *MPI-style collective operations*. These systems often involve joint efforts from machine learning, systems, and data management communities, and have been successful in making distributed training easier and more scalable.

On the theory and algorithm side, researchers have also been active in improving the performance of standard *synchronous* and *asynchronous* stochastic gradient (SG) based algorithms. Rightly noticing that a major system bottleneck is communication, researchers have proposed a range of techniques to lower the communication overhead mainly by "*relaxing*" certain aspects of the communication. Examples include (1) *communication compression* (e.g., quantization [4, 7, 73, 76], sparsification [5, 68, 70, 72], and error compensation [67]), (2) *communication decentralization* [34, 40, 42, 43, 64, 66], and (3) *communication delay* (e.g., LocalSGD [21, 44, 61, 69]) and *asynchronization* [43, 52, 60, 80, 81]. These techniques are optimized for *different workloads* and different *network conditions*. These techniques *together* hold promises to significantly decrease the communication overheads, in terms of both bandwidth and latency, or increase the tolerance to the existence of stragglers.

In this paper, we are motivated by one emerging gap between the current landscapes of systems and theory: *Despite the recent advance of distributed learning theory and algorithm on system relaxations, most, if not all, existing systems only rely on standard synchronous and*

Proceedings of the VLDB Endowment, Vol. 15, No. 4 ISSN 2150-8097.
doi:10.14778/3503585.3503590

*Equal contribution.
†Corresponding author.

Table 1: System relaxation techniques. Async algorithms let works communicate without waiting for other workers. Low precision algorithms compress the communication data. Decentralized algorithms remove the global data collection.

Alg.	Sync.	Precision	Centralization	PyTorch-DDP	Horovod	BytePS	Bagua
[38]	Sync.	Full Prec.	Centralized	✓	✓	✓	✓
[34]	Sync.	Full Prec.	Decentralized				✓
[4, 62]	Sync.	Low Prec.	Centralized	✓	✓	✓	✓
[64, 66]	Sync.	Low Prec.	Decentralized				✓
[80]	Async.	Full Prec.	Centralized			✓	✓*
[43]	Async.	Full Prec.	Decentralized				✓*
[15]	Async.	Low Prec.	Centralized				✓*
-	Async.	Low Prec.	Decentralized				

asynchronous stochastic gradient (SG) based algorithms. The main consequence is that existing systems are not taking advantage of all possible optimizations that the machine learning community has been developing, and potentially many real-world applications can be further accelerated. In this paper, we ask: *Can we further accelerate distributed learning systems with system relaxations for communications? If so, what is the right abstraction for this purpose?*

In this paper, we present Bagua, a communication library whose goal is to support state-of-the-art system relaxation techniques of distributed training. We made two technical contributions.

Our first contribution is the system design of Bagua, which provides a modular design for communications. Bagua is a natural extension of the popular parameter server and Allreduce paradigms, inspired by the challenges of directly adapting these paradigms to support algorithms in Table 1 (See Section 3.3 for details). Specifically, we provide a collection of MPI-style collective operations to facilitate communication with different precision and centralization strategies. These primitives are flexible and modular enough to support many algorithms, illustrated in Table 1. Moreover, we also develop a simple *automatic optimization framework* that speeds up algorithms implemented within the Bagua framework. The key behind this framework is automatic batching and scheduling of communications. Different from previous work such as Horovod [56] and BytePS [33], our optimization framework can be applied more widely beyond the standard SG based algorithm.

Our second contribution is an extensive empirical study centered around two hypotheses: (1) *By supporting different system relaxation techniques, Bagua is able to provide significant improvement for real-world applications and workloads with real-world infrastructure over existing systems*; and (2) *By supporting a diverse range of system relaxations, Bagua is able to provide a scalable ML training over a diverse network conditions to allow a user picking different algorithms.* To this end, we conduct a large-scale empirical study with both benchmark tasks and real-world applications running at Kwai Inc. On a cluster with up to 16 machines (128 GPUs in total, aggregated 2 petaFLOPS with Tensor Cores) we consider various network conditions following how V100 GPU machines (`p3.8xlarge`, `p3.16xlarge`, `p3dn.24xlarge`) are connected on AWS: 10Gbps, 25Gbps, and 100Gbps, with TCP/IP connections. Bagua outperforms BytePS [33], Horovod [56], and PyTorch-DDP [39] by a significant margin (up to 2× for 10Gbps and up to 1.34× for 100Gbps) across a diverse range of tasks. Moreover, we conduct a rigorous tradeoff exploration showing that different algorithms and system relaxations achieve best performance over different network conditions. This illustrates the importance of providing this diverse cohort of algorithms to an end user.

There are several limitations of the current Bagua system and we hope our efforts in building Bagua can help and inspire future research in these exciting directions. First, Bagua does not provide a principled way to help a user to *automatically* pick the most suitable system relaxations to apply. One exciting direction, after Bagua provides the support for all these algorithms, is to understand how to build a principled auto-tuning system. Second, currently Bagua only focuses on data parallelism and it is interesting future work to integrate other techniques such as model parallelism (e.g. [27, 29, 36, 49, 57, 58, 71, 75]) and pipeline parallelism (e.g., [22, 24, 41, 48]) and to understand the system abstractions.

The rest of the paper is organized as follows. We start by a brief review of data parallel training and the optimization frameworks of existing systems in Section 2, acting as both the preliminaries and related work. We discuss the design and optimization of Bagua in Section 3. We describe our experimental study in Section 4 and conclude in Section 5.

2 PRELIMINARIES AND RELATED WORK

Bagua is built on decades of research regarding distributed machine learning systems and algorithms. Plenty of them are from the database community [9, 10, 20, 28, 55]. We now summarize related work and discuss some in details to provide backgrounds and contexts. We refer the reader to [46] for the rigorous theoretical analysis of different system relaxation algorithms.

2.1 Data Parallel SG Based Algorithm

The cornerstone of distributed learning systems is the data-parallel stochastic gradient based (DP-SG) algorithms [38], which is the dominating algorithm that existing systems support and optimize for. Let D be a dataset, n is the number of workers, each worker i holds its partition of the data $D^{(i)}$ and model replica at step t: $x_i^{(t)}$. Let $g_i^{(t)}$ be the stochastic gradient on worker i at step t, a textbook DP-SG updates each local model replica, at worker i, as follows:

$$x_i^{(t+1)} = x_i^{(t)} - \gamma \sum_{j=1}^{n} g_j^{(t)}$$

where γ is the learning rate. To make this happen, all machines need to exchange their local gradients $g_i^{(t)}$, aggregate, and broadcast to all machines. Naturally, this can be implemented by the standard Allreduce communication pattern.

When there are many workers or some potential stragglers, one can extend the above algorithm into its asynchronous counterpart. Instead of using the latest gradient at iteration t, we allow the access to some staled version:

$$x_i^{(t+1)} = x_i^{(t)} - \gamma \sum_{j=1}^{n} g_j^{(\tilde{t}_j^{(i)})}$$

where $\tilde{t}_j^{(i)} \leq t$ is the previous iteration at which the gradient of worker j is computed, accessed by the worker i at iteration t. In theory, linear speedup can be achieved by async-SGD [46].

2.2 Existing Distributed Learning Systems

Distributed learning systems have attracted intensive research over the last decade. Most existing systems, e.g., DistBelief [16], Angel [32], BytePS [33], and PyTorch-DDP [39], all focus on the optimization of the DP-SG algorithm or its asynchronous counterpart. There are two fundamental questions governing the design of these systems:

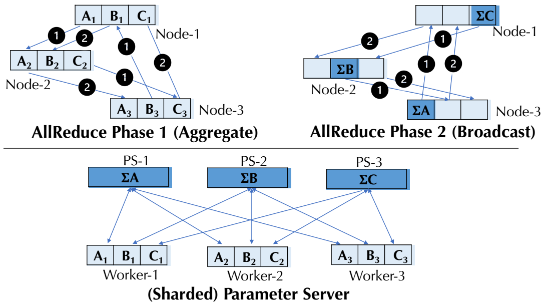


Figure 1: Illustration of Parameter Server and `Allreduce`.

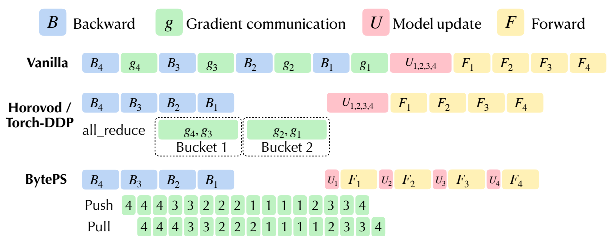


Figure 2: Communication pattern of DP-SG and how Horovod, BytePS, and PyTorch-DDP optimizes the execution for this communication pattern.

(1) *(Abstraction for Communications) How should one communicate and aggregate the gradient and model?*

(2) *(Optimizations) How should one optimize the end-to-end execution by balancing the communication and computation?*

In terms of the abstraction for communications, existing systems fall into two paradigms: parameter server (PS) [14, 16, 25, 31, 38, 78] and Allreduce [12, 30, 56, 79]. Figure 1 illustrates these two paradigms. In a *parameter server* architecture, the model can be partitioned to shards and distributed to multiple nodes (we call these nodes "parameter servers"). During the training phase, workers periodically fetch the model from PS, calculate the gradients with local data and push the gradients to the PS, while the PS aggregates the gradients and updates the parameters. With an *Allreduce* paradigm, all the workers collaborate with their neighbors for model/gradient exchanges. A ring topology [53] is often adopted by existing systems for a two-phase communication: first, the paradigm partitions the model/gradient into n chunks (where n is the number of nodes), and use n rings with different starting and ending points to aggregate n chunks; second, the aggregation result of each chunk located in different nodes is broadcast through the ring.

After deciding on which communication paradigm to use, one key design is how to hide as much communication as possible during computation. This is often the core technical component of previous systems, e.g., Horovod [56], BytePS [33], and PyTorch-DDP [39]. These systems optimize the DP-SG communication pattern by developing different ways to balance communication and computation. The key complexity roots from the fact that the training process of DP-SG consists of delicate dependencies between different layers and their own (1) forward pass, (2) backward pass, (3) gradient synchronization, and (4) model update, phases. Figure 2 (Vanilla) illustrates a naive implementation of DP-SG over a model with four layers. The system would communicate gradient (green) for each layer once its backward pass (blue) finishes, and update the model for all layers (pink) in one go once all their communications are finished. The system then starts the next forward pass (yellow).

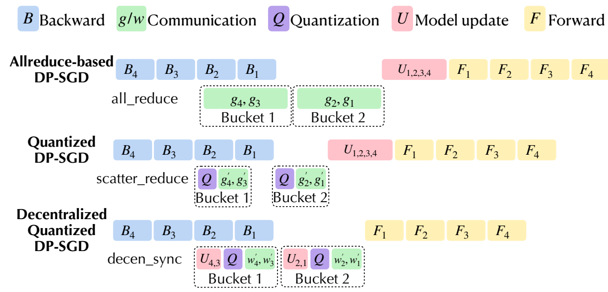


Figure 3: Communication patterns of training algorithms with system relaxations, optimized by Bagua automatically.

PyTorch-DDP and Horovod are two Allreduce-based systems, and have *specifically* optimized this pipeline by overlapping the communication (Allreduce) with the backward pass and bucketing multiple gradients into one Allreduce operation. With overlapping, the Allreduce operations can take place in parallel with the computation of gradients. The Allreduce operation is only triggered when all gradients within a bucket are ready. The intuition of bucketing is that collective communications, like Allreduce, are more efficient on large tensors. After all Allreduce operations are finished, the model will be updated by the aggregated gradients.

BytePS, following the parameter server paradigm, has optimized this pipeline in a different way. BytePS partitions each gradient into small chunks with the identical size to conduct Push/Pull. BytePS overlaps Push/Pull with both *backward and forward* pass. It has a scheduler to maintain the communication order of gradient chunks. The principle is that parameters that are blocking the execution of the next forward pass will be prioritized for communication. Therefore, the forward pass of the next iteration could possibly be overlapped with the communication of the current iteration. In terms of asynchronous DP-SG, BytePS supports it by allowing each worker updating the state of the server individually without waiting for other workers. Whereas PyTorch-DDP and Horovod do not support asynchronous communications since they rely on the Allreduce operator.

2.3 System Relaxations for Distributed DP-SG

While existing systems have been mainly focusing on synchronous and asynchronous DP-SG algorithm, the research community has developed a diverse set of techniques to further optimize for the different aspects of communications. These techniques often lead to different training algorithms, thus different communication patterns, as DP-SG. Given these differences, none of Horovod, BytePS, and PyTorch-DDP provides systematic support of these algorithms, as summarized in Table 1. The goal of Bagua is to provide a flexible abstraction to support these diverse training algorithms with an automatic performance optimization framework without assuming a specific communication pattern such as the one of DP-SG. In order to reduce communication volumes, lossy communication compression methods are introduced, such as quantization [4, 7, 73, 76], sparsification [5, 68, 70, 72], sketching [26],

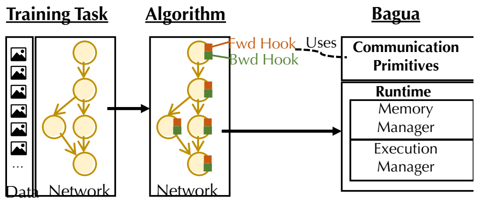


Figure 4: Overview of Bagua

and error compensation [67]). In an attempt to get rid of the latency bottleneck, decentralized communication approaches are proposed [34, 40, 42, 43, 64, 66]. Additionally, localSGD is discussed to optimize for the number of communication rounds during training [21, 44, 61, 69]. To remove the synchronization barrier, some approach applies asynchronous update methods [50, 52, 60, 80, 81]. Lastly, it is worth to mention there are approaches that combines multiple strategies listed above [6, 8, 34, 43, 65].

To illustrate the difference of communication patterns between these advanced training algorithms and vanilla DP-SG , we take the example of QSGD [4] and Decentralized Low-precision SGD [64]. Figure 3 illustrates execution pipelines and communication patterns of DP-SG, QSGD and decentralized low-precision SGD. Compared with DP-SG, the execution components of the pipeline and their dependencies can be changed in the advanced algorithms. For example, the component "Quantization" required by both algorithms doesn't even exist in the DP-SG, and the "model update" component in Decentralized low-precision SGD needs to happen *before* the communication. Since these advanced algorithms cannot fit into the DP-SG communication pattern, it is challenging for systems born for DP-SG to handle these algorithms.

3 SYSTEM DESIGN

The goal of Bagua is to support advanced training algorithms beyond DP-SG. To achieve this, we revisit the two fundamental questions governing the design of previous systems, without assuming the pattern of DP-SG:

(1) *(Abstraction for Communications) How should one communicate and aggregate the gradient and model?* **In Bagua, beyond parameter server and Allreduce, we design a collection of MPI-style collective operations to facilitate communications with different precision and centralization strategies.**
(2) *(Optimizations) How should one optimize the end-to-end execution by balancing the communication and computation?* **In Bagua, we develop a simple, but effective, automatic optimization framework which can be applied to optimize the execution of an algorithm implemented within Bagua.**

These two design decisions enable the flexibility and efficiency of Bagua— to implement a new advanced algorithm with system relaxation (e.g., 1-bit Adam [63] or Decentralized SGD [42]), in Bagua, a developer does not need to worry about manually balancing communications with computations; instead, she can specify, at a high-level, the logical semantics and Bagua will automatically optimize its execution. In this section, we first provide a high-level system overview, followed by the descriptions of these primitives and their implementations, and then the simple, but effective, optimization framework in Bagua.

3.1 System Overview

The goal of Bagua is to facilitate the development of efficient and scalable distributed training algorithms that takes advantage of system relaxations. As illustrated in Figure 4, there are three players: an *end-user*, an *optimization algorithm*, and the *Bagua runtime.*

From an end-user's perspective, using Bagua is very similar to use as PyTorch or TensorFlow for training on a single machine, with minimal changes to their existing code. The end-user should provide: (1) a neural network model that needs to train, specified as a graph in PyTorch, and (2) a stream of data examples. The end-user then specifies the training algorithm to use, e.g., QSGD [4] (training with communication compression), 1-bit Adam [63], or DecentralizedSGD [42], together with the information of the training infrastructure such as the number of machines and whether one should use MPI or NCCL for communication.

The core of Bagua is a *training algorithm*, implemented by developers using the communication primitives and abstractions provided by Bagua. An algorithm takes as input a neural network, provided by the end-user, and equips it with an algorithm-specific *communication function.* Specifically, the developer of an algorithm achieves this by registering this communication function as *hooks* at different stages of execution. One example is to register one hook after the backward computation of each layer. The communication function contains the core logic of a training algorithm, which has the following signature:

$$f((x_1, g_1)...(x_n, g_n)) \mapsto (x'_1, g'_1)...(x'_n, g'_n)$$

where (x_i, g_i) are the current model (x_i) and gradient (g_i) on the i^{th} machine and (x'_i, g'_i) are the updated model and gradient on the i^{th} machine. To implement a communication function, the developer of an algorithm assumes an MPI-like execution model. The key difference is that the developer is equipped with not only the standard communication primitives in MPI (e.g., `Allreduce`), but also a set of communication primitives provided by Bagua. These primitives support system relaxations such as compressed communications with error compensation, or decentralized communications.

When implementing the communication function in Bagua, the developer provides a *batched version* of such a function, taking as input a *set* of layers. This allows Bagua to later batch the communications automatically and optimize for its overlapping with the computations. When Bagua invokes this function, it will rearrange parameters of all layers into consecutive memory space and also pass in a *flattened* version of these layers, treat them as a single variable. An algorithm developer can decide whether her algorithm can use this flattened version to avoid conducting communication for every layer by communicating once for all the layers.

During the runtime, each invocation to the communication function (which is triggered by the registered hooks) is registered with Bagua, which equips Bagua a global view of the workload to enable automatic scheduling and batching. The key technical contribution of Bagua is to automatically apply a series of optimizations for computations and communications. To make this happen, the core of Bagua is the *execution optimizer*, which runs in two phases.

1. Profiling Phase. During the first forward/backward pass of the gradient descent computation, Bagua keeps a log of all invocations of communication functions, executes them without any optimizations. It then automatically: **(1. Bucketing)** groups layers into different buckets, whose communication will happen all at once; **(2. Flattening)** rearranges all the models and gradients of all layers in the same group into consecutive memory spaces to achieve better locality; **(3. Scheduling)** schedules when to conduct the communication of each bucket, overlapping with computations.

2. Execution Phase. For the rest forward/backward passes of the gradient decent computation, BAGUA will conduct execution over an automatically optimized version of the model. By default, BAGUA conducts one communication per bucket.

3.2 Communication Primitives

One key component of BAGUA is a collection of communication primitives. All these operators follow an execution model similar to MPI, which take as input n tensors $x_1 ... x_n$ (which can store parameter, gradient, etc.), each at a different worker, and outputs new data products $x'_1 ... x'_n$, each at a different worker:

$$op(x_1 ... x_n) \mapsto x'_1 ... x'_n$$

Centralized, Full Precision. BAGUA provides a simple primitive, `C_FP_S`, which provides the same functionality as the standard `Allreduce` operator. Specifically:

$$\texttt{C\_FP\_S}(x_1 ... x_n) \mapsto x'_1 ... x'_n \implies \forall i \in [n].\ x'_i = \sum_j x_j$$

We use this notation to express that, the effect of the `C_FP_S` operator is to calculate the sum of all local replicas, $\sum_j x_j$, and make it accessible to all workers.

Centralized, Low Precision. Communication compression has attracted intensive interests recently, given that many deep neural networks are tolerant to aggressive lossy compression of its gradient [4, 5, 7, 26, 67, 68, 70, 72, 73, 76]. BAGUA provides the `C_LP_S` primitives for this purpose. Specifically:

$$\begin{aligned}
&\texttt{C_LP_S}(x_1 ... x_n, \delta_1 ... \delta_n, \epsilon_1 ... \epsilon_n) \mapsto x'_1 ... x'_n, \delta'_1 ... \delta'_n, \epsilon'_1 ... \epsilon'_n \\
\implies & \forall i \in [n].x'_i = Q\left(\sum_j Q(x_j - \delta_j) - \epsilon_i\right) \\
& \forall i \in [n].\delta'_i = x_j - \delta_j - Q(x_j - \delta_j) \\
& \forall i \in [n].\epsilon'_i = \sum_j Q(x_j - \delta_j) - \epsilon_i - Q\left(\sum_j Q(x_j - \delta_j) - \epsilon_i\right)
\end{aligned}$$

where Q is the lossy compression function, specified by the developer and `C_LP_S` supports a general form of communication compression with error compensation [65, 67]. Note that setting δ_i and ϵ_i to `None` will disable error compensation and gives

$$\texttt{C\_LP\_S}(x_1 ... x_n, \text{None}, \text{None}) \mapsto x'_1 ... x'_n \implies \forall i \in [n].\ x'_i = Q\left(\sum_j Q(x_j)\right)$$

Intuitively, δ_i and ϵ_i keep the error caused by last iterations' compression. The convergence efficiency introduced by error compensated methods is quite robust to the compression. This technique is especially helpful when the compression function is relatively aggressive (e.g., top-K compression [5, 62]).

Decentralized, Full Precision. BAGUA also supports decentralized communication, which gets rid of the latency bottleneck for model synchronization — instead of synchronizing among all n workers in the cluster, each worker only sends the update to its neighbors. For example, according to a ring-based topology, the neighbors of a worker include its immediate left and immediate right workers in the ring. Formally, BAGUA's decentralized full precision communication primitive `D_FP_S` can be formalized as below:

$$\texttt{D\_FP\_S}(x_1 ... x_n) \mapsto x'_1 ... x'_n \implies \forall i \in [n].\ x'_i = \sum_{j \in \mathcal{N}(i)} x_j$$

where $\mathcal{N}(i)$ is the set of workers that are neighbors of worker i. Note that $\mathcal{N}(i)$ is an input to `D_FP_S`, which can be a deterministic function (e.g., fixed ring topology) or a randomized function.

Decentralized, Low Precision. BAGUA also provides the primitive `D_LP_S` for decentralized low precision communication:

$$\texttt{D\_LP\_S}(x_1 ... x_n) \mapsto x'_1 ... x'_n \implies \forall i \in [n].\ x'_i = \sum_{j \in \mathcal{N}(i)} Q(x_j)$$

Regarding the asynchronous algorithms, the current version of BAGUA does not provide any asynchronous version of these primitives, instead, it supports asynchronous algorithms using these synchronous primitives as follows. An algorithm can implement two concurrent threads, one deals with computation and another deals with communications. These two threads do not wait for each other. This provides an implementation of many asynchronous algorithms [15, 43, 80], summarized in Table 1. It can also enable implementations for LocalSGD [44] and model averaging [74]. It is interesting to further explore the benefits of providing asynchronous version of primitives, which we leave as future work.

3.3 Comparisons with PS and Allreduce

We see BAGUA as a *natural extension* of two existing paradigms, i.e., parameter server and `Allreduce`. The design of BAGUA's communication primitives is inspired by challenges that we faced when trying to implement all eight different communication patterns in Table 1 when directly using these paradigms, discussed as follows.

Parameter Server. Considering the centralized, low precision communication pattern (`C_LP_S`), we found it could be unnatural to implement such a pattern by directly using the `put/get` abstraction provided by a parameter server. The fundamental challenge is that the error compensation step requires us to keep a state (to remember the error) on the server side and continue to conduct accumulation and quantization for each `get` request. The `C_LP_S` primitive is an extension of the parameter server abstraction for this purpose. A more fundamental problem is to support decentralized algorithms (`D_FP_S`) using the parameter server abstraction — we have to explicitly specify the communication topology. The `D_FP_S` primitive is an extension for this purpose.

`Allreduce`. Similarly, using `AllReduce` operators in MPI also faces challenges when implementing a centralized, low precision communication pattern. It is not clear how to keep track of the quantization error made by `Allreduce` at each communication step and compensate it in the next iteration. Similarly, it can also be unnatural to support the decentralized communication pattern using existing operators in MPI.

3.4 Implementations of Primitives

For the centralized primitives, BAGUA adopts a ScatterReduce communication pattern [79]. Specially, the target tensor is divided into n partitions, where n is the number of workers. The i-th worker is responsible for aggregating the i-th partition. Since the underlying communication library NCCL does not provide a ScatterReduce primitive, we implement this primitive using the basic *send* and *recv* NCCL operators. Each worker 1) partitions local tensor, 2) sends partitions to corresponding workers, 3) receives responsible partitions from other workers, 4) merges received partitions, and 5) sends merged partition to other workers. ScatterReduce communication pattern can take advantage of the aggregated bandwidth of all workers (like Allreduce), and support compression techniques (unlike Allreduce). The low precision primitive `C_LP_S` leverages the ScatterReduce communication to incorporate two phases of compression. Note that, the compression and decompression procedures can be combined with error compensation technique to reduce information loss (see semantics in Section 3.2).

Unlike centralized training, workers in decentralized training only communicates with one or a few peers. Bagua engineers two mechanisms to allocate peers — *ring* and *random*. The *ring* strategy gives successive ranks to workers and organizes all workers as a ring. Alternatively, the *random* strategy randomly chooses a peer for each worker. The low precision primitive `D_LP_S` uses the same peer selection and communication procedure as `D_FP_S`. The difference is that `D_LP_S` uses the compression function Q to compress the tensor.

3.5 Bagua Optimization Framework

The central component of Bagua is its *execution optimizer*. The goal of Bagua's execution optimizer is to automatically schedule and optimize the computations and communications during the training. We explore the following techniques in Bagua.

Overlapping communication and computation is one central optimization to speedup distributed DP-SG. Bagua automatically analyzes the computation graph that includes the in-place tensor operations and tensor communication primitives. Compared with existing systems, Bagua considers more sophisticated scheduling. In vanilla DP-SG, the optimization can only hide the Allreduce communications inside the gradient computation, Bagua is responsible for scheduling additional elements, such as compression/decompression and the model update computations specified by the optimization algorithms (E.g., Figure 3).

In order to boost the efficiency of communication and parallel computation, fusing small tensors into buckets is an essential step — frequently calling the communication primitives to transfer small fragments of parameters is far from ideal in terms of fully utilizing the network bandwidth. Although the bucketing trick is adopted in both Horovod and PyTorch-DDP, their schema simply considers the Allreduce operation as the cost in the heuristic. By contrast, Bagua needs to consider beyond the Allreduce. Once we split the computation graph into buckets, Bagua conducts fusion over the buckets. Bagua would carefully align parameters within a bucket into a continuous memory space. Then this flatten view of the parameters is leveraged for all the executions. For example, the low-precision compression/decompression lambda is directly applied over a flatten view of the bucket instead of individual tensors; the SG based optimizer for model update is also conducted at the level of buckets (Apex [1] also uses a similar optimization). This flatten view can utilize the parallelism of GPUs more effectively.

Last but not least, the communication of Bagua can be conducted hierarchically. This is particularly useful when dealing with the heterogeneity in network connections, e.g., the bandwidth between GPUs within a server is much higher than the bandwidth between servers. Therefore, Bagua communicates hierarchically in two levels: intra-node and inter-node, and optimizes the communication primitives based on this abstraction. For example, the centralized low-precision primitive (`C_LP_S`) can be optimized as first aggregating tensors within the node *without* compression, then performing inter-node aggregation over the leader workers *with* compression, and finally letting each leader worker broadcast aggregated data within the node. Notice that this optimization can potentially change the semantics of the communication primitives.

4 EVALUATION

We conduct extensive experimental study around three hypotheses:

- *Bagua is able to provide significant performance improvements over state-of-the-art systems in terms of end-to-end training time and scalability, over realistic industrial-scale infrastructure.*
- *Different algorithms that Bagua support provide benefits for different models and datasets under different network conditions. It is thus important for Bagua to support all these algorithms.*
- *Bagua's automatic execution optimizer effectively optimizes the execution of various distributed training algorithms.*

4.1 Experimental Setting

Infrastructure. All experiments are conducted on 16-GPU instances, each of which is equipped with 8 NVIDIA V100 32GB GPUs interconnected by NVLink. We consider three different network conditions following how V100 GPU machines (`p3.8xlarge`, `p3.16xlarge`, `p3dn.24xlarge`) are connected on AWS: 10Gbps, 25Gbps, and 100Gbps, with TCP/IP connections. The default bandwidth we are using is 100Gbps without specification.

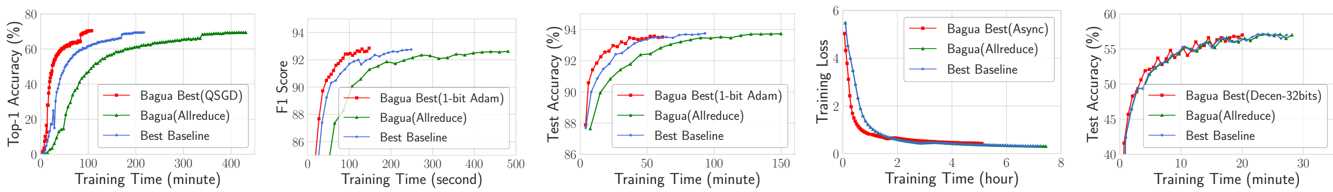
Competing Systems. We compare the performance of Bagua with three SOTA systems. **PyTorch-DDP** [39], Pytorch's default solution of distributed data parallelism learning. **Horovod** [56], a distributed learning framework developed by Uber. **BytePS** [33], a distributed learning platform developed by ByteDance. Both PyTorch-DDP and Horovod relies on MPI Allreduce for communication while BytePS uses parameter servers. Horovod and PyTorch-DDP also supports **fp16 gradient compression** via the fp16 support in NVIDIA NCCL, which we also compare with.

Datasets and Tasks. We use five learning tasks, covering both standard benchmarks and production datasets at Kwai Inc: **(1) Image**: (ImageNet [17], VGG16 [59]); **(2) Text**: (SQuAD [54], BERT-LARGE finetune [18]); **(3) Text**: (Kwai Dataset, BERT-BASE finetune [18]); **(4) Speech**: (AISHELL-2 [19], Transformer); **(5) Image+Text**: (Kwai Dataset, LSTM [23]+AlexNet [35]). Table 2 summarizes the model size and FLOPs.

Table 2: Model Characteristics

	VGG16	BERT-LARGE	BERT-BASE	Transformer	LSTM+AlexNet
# Parameters	138.3M	302.2M	85.6M	66.5M	126.8M
# FLOPs	31G	232G	22G	145G	97.12G

Bagua Algorithms. We implemented six algorithms in Bagua. **Allreduce**, the standard DP-SG algorithm, implemented with `C_FP_S` primitive. **QSGD** [4], a quantized (8-bit) DP-SG algorithm, implemented with `C_LP_S` primitive without error compensation. **1-bit Adam** [63], a quantized (1-bit) distributed learning algorithm, implemented with by `C_LP_S` primitive with error compensation. **Decen-32bits**, a decentralized training algorithm with the random



(a) VGG16 (b) BERT-LARGE Finetune (c) BERT-BASE Finetune (d) Transformer (e) LSTM+AlexNet

Figure 5: End-to-end performance of Bagua and the best competing baseline. Over all five tasks, Horovod-16bits is the *best* of {Torch-DDP, Horovod-32bits, Horovod-16bits, BytePS}. We show the performance of Bagua Allreduce and the optimal Bagua algorithm selected for each task. The bandwidth of inter-machine network is 10Gbps.

probing method to exchange the model parameters in each iteration, implemented with D_FP_S. **Decen-8bits** [64], a ring-based decentralized training algorithm with quantization, implemented with D_LP_S. **Async**, asynchronous centralized DP-SG.

4.2 End-to-end Comparison with SOTAs

We first conduct end-to-end experiments over the network with 10Gbps bandwidth, following the setup of V100 instance `p3.8xlarge` on AWS. In Figure 5, we compare Bagua with Horovod-16bits, which is the best of all the baseline systems in this setting. We report the results of Bagua Allreduce and the best algorithm in Bagua we select for each task by considering both the accuracy and efficiency. As we can see, Bagua can be up to 2× faster than the best competing system, while still guaranteeing the same model accuracy. Bagua Allreduce is the slowest one in this case because it has the largest amount of data transition. These results also reflect the effectiveness and necessity of supporting various algorithms.

There are multiple reasons behind the speedup of Bagua. The most direct one comes from the communication-efficient algorithms. For example, Bagua (QSGD-8bits and 1-bit Adam) can compress the data communication much more aggressively than Horovod-16bits, Bagua (Decentralized 32bits and 8bits) could have much less communication connections than Allreduce-based systems, and Bagua (Async) can make the communication totally independent of the computation. Therefore, the overall communication overhead is highly reduced by Bagua. Besides, Bagua has a well-optimized execution pipeline for the computation and communication, including tensor flattening and bucketing, memory management, overlapping and so on. All the algorithms implemented in Bagua are automatically benefiting from these system optimizations.

4.3 Trade-off of Bagua Algorithms

By supporting a diverse collection of algorithms, Bagua provides users flexibility to accommodate different tasks, network conditions (in terms of latency and throughput), and worker heterogeneities. As we will see, there is no one-size-fits-all algorithm in Bagua that is always optimal for all these situations. Instead, we see an interesting trade-off space of Bagua algorithms.

Convergence. *The convergence behavior of different algorithms heavily depends on the tasks; thus, it is important to support a diverse cohort in Bagua.* Figure 6 illustrates the convergence behavior of different algorithms. Taking VGG16 as an example and treating Bagua (Allreduce) as the baseline algorithm, QSGD and Async can

Table 3: Epoch time (s) with one straggler GPU

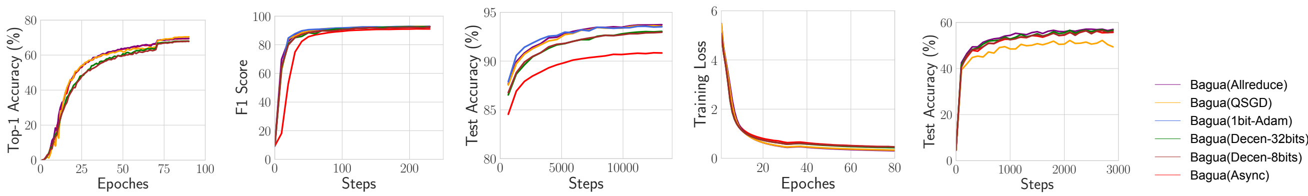
	VGG16	Bert-large	Bert-base	Transformer	LSTM+AlexNet
Bagua (Async)	48	57	421	185	128
Bagua (1-bit Adam)	131	154	762	411	313
Bagua (QSGD)	132	154	908	420	316
Bagua (Decen-32bits)	138	164	885	452	324
Bagua (Decen-8bits)	136	165	800	455	320

have almost the same convergence rate, whereas Decen-32bits and Decen-8bits show some drop of accuracy. 1-bit Adam algorithm can't converge on VGG16 and the training loss explodes after a few epochs because an important assumption of 1-bit Adam regarding the gradients variance cannot hold in this task. For BERT-LARGE, most algorithms can converge in a similar rate as Allreduce, except Async algorithm that suffers an obvious gap. For LSTM+AlexNet, Decen-32bits, Decen-8bits and Async can converge as Allreduce does, the performance of QSGD is degraded, and 1-bit Adam diverges again. The above results verify that different training algorithms show diverse convergence behaviours and there is NO algorithm that can beat others across all workloads. Unfortunately, given a specific task, Bagua currently cannot choose the best algorithm for users. Although a prior work [45] has theoretically given the convergence bounds of different training algorithms, it is still unsolved which algorithm of them achieves the best empirical convergence rate without actually running them. Later, we will provide some guidelines for uses based on our empirical study.

Network Conditions. Among the set of algorithms that have similar convergence behavior as Allreduce, their relative performance is governed by the underlying network conditions: latency and bandwidth. We vary these two factors and illustrate the epoch time in Figure 7 (We show BERT-LARGE, but other tasks have similar profile). Algorithms that conduct communication compression outperforms others when the bandwidth is relatively low; whereas decentralized algorithms outperform others when the latency is relatively high. We see when the network gets slower, the gap between Bagua and other systems becomes even larger.

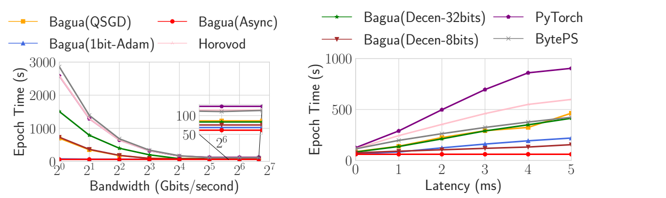
Worker Heterogeneity. We also simulate a heterogeneous cluster by manually degrading the *Applications Clocks* of one GPU. Specifically, we set the frequency of *Graphics* from 1290MHz to 585MHz. As shown in Table 3, when there are stragglers in the system, asynchronous algorithms outperform a synchronous one in terms of epoch time, which is also consistent with previous observations [51].

Insights and Guidlines. These results justify the fundamental motivation of Bagua: at the algorithmic level, there is no algorithm



(a) VGG16 (b) BERT-LARGE Finetune (c) BERT-BASE Finetune (d) Transformer (e) LSTM+AlexNet

Figure 6: Convergence of different algorithms



(a) Bandwidth (b) Latency

Figure 7: Epoch time under different network conditions, BERT-LARGE Finetune

Table 4: Epoch time (s) of the *centralized full-precision synchronized* algorithm of different systems.

	VGG16	Bert-large	Bert-base	Transformer	AlexNet+LSTM
Bagua AllReduce	105	114	510	318	168
PyTorch-DDP	106	116	521	341	171
Horovod	107	112	550	343	177
BytePS	170	114	548	340	224

that can serve as a sliver bullet for all the distributed training tasks; as so, it is essential for a distributed learning system like Bagua to be able to effectively fill the gap between the communication primitives defined by the infrastructure and the system relaxation demanded by various distributed learning algorithms. A further step beyond this empirical study is to automatically choose the optimal training algorithm for a given task and specific network condition. However, this interesting topic is orthogonal to our work and we will explore it in the future. Currently, we can provide users some empirical guidelines to choose the algorithm. For example:

- For the low bandwidth network, compressed algorithms (QSGD, 1-bit Adam, Decen-8bits) are likely to be better.
- For the high latency network, decentralized algorithms (Decen-32bits, Decen-8bits) are likely to be better.
- If the original optimizer is SGD, we recommend QSGD. While if the original optimizer is Adam, we recommend 1-bit Adam.
- If the communication/computation ratio is quite small, Async algorithm could be your choice to further reduce the communication overhead.

4.4 Ablation Study of System Optimizations

We now validate the effectiveness of the Bagua optimization framework. As described in Section 3.5, the optimization framework consists of three main optimizations: **O**: Overlapping between the training computation and Bagua execution; **F**: Fusion and Flattening of tensors. **H**: Hierarchical Communications.

We first apply Bagua to the standard DP-SG algorithm and compare with PyTorch-DDP, Horovod, and BytePS, as illustrated

Table 5: Epoch time (s) with different system optimizations

	VGG16	Bert-large	Bert-base	Transformer	LSTM+AlexNet
O=1,F=1,H=1	74	67	369	185	148
O=0,F=1,H=1	88	70	395	185	163
O=1,F=0,H=1	117	148	617	185	210
O=1,F=1,H=0	510	128	572	185	146

in Table 4. Different from these systems that manually optimize *specifically* for DP-SG, Bagua automatically optimizes for an algorithm that is implemented within its framework. We see that Bagua achieves similar, and sometimes better, performance, illustrating the effectiveness of Bagua's optimization framework.

Second, we show that all three optimizations are crucial for the end-to-end performance of Bagua, and the benefits of them can vary significantly from task to task. We conduct an ablation study and Table 5 illustrates the result (X=0 means the optimization X is tuned off). We see different optimizations are important for different workloads. In principle, **O** can overlap as much operations as possible, **F** can make communications of small tensor more efficient, and **H** accelerates intra-node GPU communications. These intuitions can be observed from the empirical results. For communication intensive workloads (e.g., VGG-16), hierarchical communication improves the performance significantly. For models with many small tensors (e.g., BERT-LARGE) and decentralized communication patterns (e.g., LSTM+AlexNet), fusion and overlapping play a larger role. An exception is the asynchronous algorithm (Transformer) because its communication is completely independent with the computation, therefore, it doesn't effect the epoch time.

5 CONCLUSION

We propose Bagua, a communication framework whose design goal is to support various distributed training algorithms with system relaxations, powered by a new system design and a simple but effective optimization framework. We conduct empirical study to illustrate the end-to-end performance of Bagua and to provide a systematic trandeoff study of different training algorithms.

ACKNOWLEDGMENTS

CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628, and 197485), Innosuisse/SNF BRIDGE Discovery (Project Number 40B2-0_187132), European Union Horizon 2020 Research and Innovation Programme (DAPHNE, 957407), Botnar Research Centre for Child Health, Swiss Data Science Center, Alibaba, Cisco, eBay, Google Focused Research Awards, Kuaishou Inc., Oracle Labs, Zurich Insurance, and the Department of Computer Science at ETH Zurich.

REFERENCES

[1] [n.d.]. Apex. https://nvidia.github.io/apex/optimizers.html.

[2] [n.d.]. NCCL. https://developer.nvidia.com/nccl.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 265–283.

[4] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2016. QSGD: Communication-efficient SGD via gradient quantization and encoding. *arXiv preprint arXiv:1610.02132* (2016).

[5] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Sarit Khirirat, Nikola Konstantinov, and Cédric Renggli. 2018. The convergence of sparsified gradient methods. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 5977–5987.

[6] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. 2019. Qsparse-local-SGD: Distributed SGD with quantization, sparsification, and local computations. *arXiv preprint arXiv:1906.02367* (2019).

[7] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. 2018. signSGD: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*. PMLR, 560–569.

[8] Aleksandr Beznosikov, Samuel Horváth, Peter Richtárik, and Mher Safaryan. 2020. On biased compression for distributed learning. *arXiv preprint arXiv:2002.12410* (2020).

[9] Matthias Boehm, Michael W Dusenberry, Deron Eriksson, Alexandre V Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R Reiss, Prithviraj Sen, Arvind C Surve, et al. 2016. Systemml: Declarative machine learning on spark. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1425–1436.

[10] Matthias Boehm, Shirish Tatikonda, Berthold Reinwald, Prithviraj Sen, Yuanyuan Tian, Douglas R Burdick, and Shivakumar Vaithyanathan. 2014. Hybrid parallelization strategies for large-scale machine learning in systemml. *Proceedings of the VLDB Endowment* 7, 7 (2014), 553–564.

[11] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[12] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[13] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project adam: Building an efficient and scalable deep learning training system. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 571–582.

[14] Wei Dai, Abhimanu Kumar, Jinliang Wei, Qirong Ho, Garth Gibson, and Eric P Xing. 2015. High-performance distributed ML at scale through parameter server consistency models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[15] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. 2017. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 561–574.

[16] Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, et al. 2012. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems-Volume 1*. 1223–1231.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

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

[19] Jiayu Du, Xingyu Na, Xuechen Liu, and Hui Bu. 2018. Aishell-2: Transforming mandarin asr research into industrial scale. *arXiv preprint arXiv:1808.10583* (2018).

[20] Arash Fard, Anh Le, George Larionov, Waqas Dhillon, and Chuck Bear. 2020. Vertica-ML: Distributed Machine Learning in Vertica Database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 755–768.

[21] Farzin Haddadpour, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck R Cadambe. 2019. Local sgd with periodic averaging: Tighter analysis and adaptive synchronization. *arXiv preprint arXiv:1910.13598* (2019).

[22] Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. 2021. PipeTransformer: Automated Elastic Pipelining for Distributed Training of Transformers. *arXiv preprint arXiv:2102.03161* (2021).

[23] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[24] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism.. In *NeurIPS*.

[25] Yuzhen Huang, Tatiana Jin, Yidi Wu, Zhenkun Cai, Xiao Yan, Fan Yang, Jinfeng Li, Yuying Guo, and James Cheng. 2018. Flexps: Flexible parallelism control in parameter server architecture. *Proceedings of the VLDB Endowment* 11, 5 (2018), 566–579.

[26] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Ion Stoica, Raman Arora, et al. 2019. Communication-efficient distributed sgd with sketching. In *Advances in Neural Information Processing Systems*. 13144–13154.

[27] Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J Gao. 2019. Declarative recursive computation on an RDBMS: or, why you should use a database for distributed machine learning. *Proceedings of the VLDB Endowment* 12, 7 (2019), 822–835.

[28] Matthias Jasny, Tobias Ziegler, Tim Kraska, Uwe Roehm, and Carsten Binnig. 2020. DB4ML-An In-Memory Database Kernel with Machine Learning Support. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 159–173.

[29] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Data and Model Parallelism for Deep Neural Networks. *SysML 2019* (2019).

[30] Jiawei Jiang, Bin Cui, Ce Zhang, and Fangcheng Fu. 2018. Dimboost: Boosting gradient boosting decision tree to higher dimensions. In *Proceedings of the 2018 International Conference on Management of Data*. 1363–1376.

[31] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*. 463–478.

[32] Jie Jiang, Lele Yu, Jiawei Jiang, Yuhong Liu, and Bin Cui. 2018. Angel: a new large-scale machine learning system. *National Science Review* 5, 2 (2018), 216–236.

[33] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed {DNN} Training in Heterogeneous GPU/CPU Clusters. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 463–479.

[34] Anastasia Koloskova, Sebastian Stich, and Martin Jaggi. 2019. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *International Conference on Machine Learning*. PMLR, 3478–3487.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.

[36] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).

[37] Hao Li, Asim Kadav, Erik Kruus, and Cristian Ungureanu. 2015. Malt: distributed data-parallelism for existing ml applications. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–16.

[38] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 583–598.

[39] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. [n.d.]. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proceedings of the VLDB Endowment* 13, 12 ([n. d.]).

[40] Youjie Li, Mingchao Yu, Songze Li, Salman Avestimehr, Nam Sung Kim, and Alexander Schwing. 2018. Pipe-SGD: a decentralized pipelined SGD framework for distributed deep net training. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 8056–8067.

[41] Zhuohan Li, Siyuan Zhuang, Shiyuan Guo, Danyang Zhuo, Hao Zhang, Dawn Song, and Ion Stoica. 2021. TeraPipe: Token-Level Pipeline Parallelism for Training Large-Scale Language Models. *arXiv preprint arXiv:2102.07988* (2021).

[42] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 5336–5346.

[43] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. 2018. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*. PMLR, 3043–3052.

[44] Tao Lin, Sebastian U Stich, Kumar Kshitij Patel, and Martin Jaggi. 2019. Don't Use Large Mini-batches, Use Local SGD. In *International Conference on Learning Representations*.

[45] Ji Liu and Ce Zhang. 2021. Distributed learning systems with first-order methods. *arXiv preprint arXiv:2104.05245* (2021).

[46] Ji Liu, Ce Zhang, et al. 2020. Distributed Learning Systems with First-Order Methods. *Foundations and Trends® in Databases* 9, 1 (2020), 1–100.

[47] Ruben Mayer and Hans-Arno Jacobsen. 2020. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Computing Surveys (CSUR)* 53, 1 (2020), 1–37.

[48] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of*

the 27th ACM Symposium on Operating Systems Principles. 1–15.

[49] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2020. Memory-efficient pipeline-parallel dnn training. *arXiv preprint arXiv:2006.09503* (2020).

[50] Lam Nguyen, Phuong Ha Nguyen, Marten Dijk, Peter Richtárik, Katya Scheinberg, and Martin Takác. 2018. SGD and Hogwild! convergence without the bounded gradients assumption. In *International Conference on Machine Learning*. PMLR, 3750–3758.

[51] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J Wright. 2011. HOGWILD! a lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*. 693–701.

[52] Hao Peng, Shandian Zhe, Xiao Zhang, and Yuan Qi. 2017. Asynchronous distributed variational Gaussian process for regression. In *International Conference on Machine Learning*. PMLR, 2788–2797.

[53] Rolf Rabenseifner. 2004. Optimization of collective reduction operations. In *International Conference on Computational Science*. Springer, 1–9.

[54] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[55] Sandeep Singh Sandha, Wellington Cabrera, Mohammed Al-Kateb, Sanjay Nair, and Mani Srivastava. 2019. In-database distributed machine learning: demonstration using Teradata SQL engine. *Proceedings of the VLDB Endowment* 12, 12 (2019), 1854–1857.

[56] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[57] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. 2018. Mesh-TensorFlow: deep learning for supercomputers. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 10435–10444.

[58] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[59] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[60] Umut Simsekli, Cagatay Yildiz, Than Huy Nguyen, Taylan Cemgil, and Gael Richard. 2018. Asynchronous stochastic quasi-Newton MCMC for non-convex optimization. In *International Conference on Machine Learning*. PMLR, 4674–4683.

[61] Sebastian U Stich. 2018. Local SGD Converges Fast and Communicates Little. In *International Conference on Learning Representations*.

[62] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. 2018. Sparsified SGD with Memory. *Advances in Neural Information Processing Systems* 31 (2018), 4447–4458.

[63] Hanlin Tang, Shaoduo Gan, Ammar Ahmad Awan, Samyam Rajbhandari, Conglong Li, Xiangru Lian, Ji Liu, Ce Zhang, and Yuxiong He. 2021. 1-bit Adam: Communication Efficient Large-Scale Training with Adam's Convergence Speed. *arXiv preprint arXiv:2102.02888* (2021).

[64] Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. 2018. Communication compression for decentralized training. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 7663–7673.

[65] Hanlin Tang, Xiangru Lian, Shuang Qiu, Lei Yuan, Ce Zhang, Tong Zhang, and Ji Liu. 2019. Deepsqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. *arXiv preprint arXiv:1907.07346* (2019).

[66] Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. 2018. D2: Decentralized training over decentralized data. In *International Conference on Machine Learning*. PMLR, 4848–4856.

[67] Hanlin Tang, Chen Yu, Xiangru Lian, Tong Zhang, and Ji Liu. 2019. Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. In *International Conference on Machine Learning*. PMLR, 6155–6165.

[68] Hongyi Wang, Scott Sievert, Zachary Charles, Shengchao Liu, Stephen Wright, and Dimitris Papailiopoulos. 2018. ATOMO: communication-efficient learning via atomic sparsification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 9872–9883.

[69] Jianyu Wang and Gauri Joshi. 2019. Adaptive Communication Strategies to Achieve the Best Error-Runtime Trade-off in Local-update SGD. In *Systems and Machine Learning (SysML) Conference*.

[70] Jialei Wang, Mladen Kolar, Nathan Srebro, and Tong Zhang. 2017. Efficient distributed learning with sparsity. In *International Conference on Machine Learning*. PMLR, 3636–3645.

[71] Minjie Wang, Chien-chin Huang, and Jinyang Li. 2019. Supporting very large models using automatic dataflow graph partitioning. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–17.

[72] J Wangni, J Liu, J Wang, and T Zhang. 2018. Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems* 31 (2018), 1299.

[73] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: ternary gradients to reduce communication in distributed deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1508–1518.

[74] Hao Yu, Sen Yang, and Shenghuo Zhu. 2019. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 5693–5700.

[75] Binhang Yuan, Dimitrije Jankov, Jia Zou, Yuxin Tang, Daniel Bourgeois, and Chris Jermaine. 2021. Tensor Relational Algebra for Distributed Machine Learning System Design. *Proc. VLDB Endow.* 14, 8 (2021), 1338–1350.

[76] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. 2017. Zipml: Training linear models with end-to-end low precision, and a little bit of deep learning. In *International Conference on Machine Learning*. PMLR, 4035–4043.

[77] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. 2017. Poseidon: An efficient communication architecture for distributed deep learning on {GPU} clusters. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*. 181–193.

[78] Zhipeng Zhang, Bin Cui, Yingxia Shao, Lele Yu, Jiawei Jiang, and Xupeng Miao. 2019. Ps2: Parameter server on spark. In *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data*. 376–388.

[79] Zhipeng Zhang, Jiawei Jiang, Wentao Wu, Ce Zhang, Lele Yu, and Bin Cui. 2019. Mllib*: Fast training of glms using spark mllib. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 1778–1789.

[80] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. 2017. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning*. PMLR, 4120–4129.

[81] Zhengyuan Zhou, Panayotis Mertikopoulos, Nicholas Bambos, Peter Glynn, Yinyu Ye, Li-Jia Li, and Li Fei-Fei. 2018. Distributed asynchronous optimization with unbounded delays: How slow can you go?. In *International Conference on Machine Learning*. PMLR, 5970–5979.

[82] Yongqiang Zou, Xing Jin, Yi Li, Zhimao Guo, Eryu Wang, and Bin Xiao. 2014. Mariana: Tencent deep learning platform and its applications. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1772–1777.