Communication Efficient Asynchronous Stochastic Gradient Descent

Youssef Ahmed[1], Arnob Ghosh[2], Chih-Chun Wang[3], and Ness B. Shroff[1,4]

[1]Department of ECE, The Ohio State University, Columbus, OH, USA
[2]Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA
[3]Elmore Family School of ECE, Purdue University, West Lafayette, IN, USA
[4]Department of CSE, The Ohio State University, Columbus, OH, USA
Emails: ahmed.943@osu.edu, arnob.ghosh@njit.edu, chihw@purdue.edu, shroff.11@osu.edu

***Abstract*—In this paper, we address the challenges of asynchronous gradient descent in distributed learning environments, particularly focusing on addressing the challenges of stale gradients and the need for extensive communication resources. We develop a novel communication efficient framework that incorporates a gradient evaluation algorithm to assess and utilize delayed gradients based on their quality, ensuring efficient and effective model updates while significantly reducing communication overhead. Our proposed algorithm requires agents to only send the norm of the gradients rather than the computed gradient. The server then decides whether to accept the gradient if the ratio between the norm of the gradient and the distance between the global model parameter and the local model parameter exceeds a certain threshold. With the proper choice of the threshold, we show that the convergence rate achieves the same order as the synchronous stochastic gradient without depending on the staleness value unlike most of the existing works. Given the computational complexity of the initial algorithm, we introduce a simplified variant that prioritizes the practical applicability without compromising on the convergence rates. Our simulations demonstrate that our proposed algorithms outperform existing state-of-the-art methods, offering improved convergence rates, stability, accuracy, and resource consumption.**

I. Introduction

Distributed Stochastic Gradient Descent (SGD) serves as the cornerstone for distributed optimization of machine learning models, with its synchronous and asynchronous variants offering different trade-offs in terms of computational efficiency and convergence properties. For example, synchronous SGD requires that all the workers (i.e., the distributed machines) compute gradients on the same model parameters. Naturally, this requires a large amount of communication overhead for synchronization especially for a large number of distributed machines. On the other hand, Asynchronous SGD (ASGD) relaxes the above limitation and allows that workers may compute gradients on different model parameters, thereby offering greater scalability and resource utilization. This makes ASGD suitable for large-scale distributed systems and heterogeneous computing environments where different workers may have varying speeds and capabilities. However, this advantage comes at the cost of introducing stale gradients— gradient updates computed using outdated versions of the model parameters. These stale gradients occur because some workers may use older parameters to calculate their gradients [1], [2]. The stale gradient problem is a significant challenge in ASGD, as it degrades convergence properties and can lead to resource inefficiencies due to instability and increased computation requirements [1], [3]–[8].

The stale gradients indeed affect the theoretical analysis of convergence rates of all the proposed ASGD algorithms [4], [7], [9]–[16]. In particular, all the above papers assume bounded staleness assumptions and the convergence rate depends on the staleness parameters which can be large in practice. Moreover, the advent of federated learning has further complicated the stale gradient issue, introducing additional layers of complexity such as device heterogeneity and Non-IID data distributions across clients. Federated learning's promise of collaborative model training without compromising data privacy necessitates innovative solutions to the staleness challenge, ensuring that the global model converges effectively despite asynchronous updates from a diverse set of clients [17], [18].

In addition to the convergence issue, the staleness poses another challenge, which is the potential for *resource inflation due to instability* [3]. While asynchronous SGD is very efficient at utilizing computational resources by allowing parallel updates, this efficiency can be compromised due to increased demands placed on the system when instability arises. Specifically, as the objective value fluctuates due to stale gradients, it may initially decrease but then increase again, necessitating additional iterations to regain stability. In other words, using delayed gradients to update the model can sometimes deteriorate the objective value (i.e., increase the learning loss), requiring additional resources (e.g., computation power and communication rounds) to compensate for updating the parameters along the wrong direction of the gradient.

While the presence of delayed gradients in asynchronous SGD has *traditionally* been viewed as a hurdle for stable and efficient learning, recent insights suggest a more nuanced perspective. Specifically, under certain conditions, these delayed updates can introduce a form of momentum into the learning process, akin to the momentum term used in optimization algorithms to accelerate convergence [19]. This phenomenon, explored in depth in works such as [9], hinges on the distribution of staleness across updates. When delays follow a specific statistical pattern, such as a geometric distribution, the cumulative effect of asynchronously applied gradients can mimic the behavior of explicit momentum, thereby enhancing the algorithm's performance. This counter-intuitive finding reveals that not all stale gradients are detrimental; indeed, empirical results suggest that under the right circumstances, they may be utilized to accelerate the learning process. Recently, [20] proposed an algorithm that scales down the learning rate as the staleness increases. In particular, as the staleness value increases, such a value contribute very minutely towards the parameter update. [20] showed that such an approach achieves a convergence rate devoid of any staleness parameter. However, the above algorithm requires communication of all the computed gradients between the worker and the server which can *overcrowd* the underlying bandwidth even when the contribution of the gradients might be small as they have high stale value. More importantly, perhaps, a gradient with a higher staleness value can contribute towards convergence more compared to a gradient with a lower staleness value. However, [20] puts a lower weight on the gradients with a higher stale value. Hence, the empirical results indicate that the convergence performance is not comparable to the synchronous SGD. This raises the question: *Can we develop an algorithm where stale gradients can be utilized effectively to improve performance in ASGD? In particular, can we develop a resource efficient ASGD algorithm* with provable *convergence rates that is not hindered by the staleness?*

In this paper, we address the question by developing a novel ASGD algorithm. The key component of our algorithm is that the server only accepts the stale-gradients when the ratio of the norm of the stale-gradient, and the distance between the current model parameter and the outdated model parameter is above a certain threshold which we denote as 'high quality gradient'. Further, the worker does not need to send all the computed gradients, rather, sending the norm of the gradient is enough which is *far less resource intensive*. We show that by carefully designing the threshold, we can *achieve* the same order of convergence as that of the synchronous SGD. Since we only accept those gradients that contribute towards the convergence, our approach requires smaller communication resource. Here are our contributions in details:

1) We introduce an innovative quality metric that assesses the usefulness of delayed gradients. This metric determines whether to use a gradient for model updates based on its norm (rather than the entire gradient), and the amount of change between the delayed parameter and the recent parameter (rather than the staleness). Namely, instead of using the time stamps to decide whether to "accept" a recently computed gradient or not, we directly examine the "quality" of the computed gradient instead. Our approach may accept a gradient even when its time stamp is older if it is of 'high quality'. Further, if we are accepting a gradient we are not down-scaling the learning rate as existing approaches fully utilizing their values. By directly examining the quality of the update, not just the timeliness, this approach reduces the communication overhead, making the process more communication-efficient and staleness independent.
2) Based on the quality metric, we have developed a novel variation of the ASGD algorithm called Quality-Aware Asynchronous Stochastic Gradient Descent (QASY). This algorithm enhances the communication efficiency of ASGD by selectively utilizing useful delayed gradients for model updates. Additionally, QASY assesses the usefulness of gradients stored from previous iterations and incorporates only the beneficial ones in the model update.
3) We provide a detailed convergence analysis for QASY, showing that the convergence rate and learning rate are independent of the staleness value and similar to the convergence rate of the synchronous SGD. In particular, *we show that one can achieve convergence rate of $O(1/\sqrt{T})$ (where T is the number of iterations) in asynchronous SGD devoid of any staleness parameter*. Hence, we bridge the gap between the the synchronous SGD and the asynchronous SGD in terms of convergence rate using smaller communication resources. Also, we propose simplified version of QASY, Simplified Quality-Aware Asynchronous Stochastic Gradient Descent (s-QASY), that has the same performance guarantees. It streamlines the process by focusing solely on the utilization of delayed gradients. This variant reduces the computational overhead further while retaining the core benefits of our gradient management strategy.
4) Through simulations, we validate the effectiveness of our proposed algorithms, comparing them with state-of-the-art methods in asynchronous gradient descent. Our results confirm the superiority of our approaches in terms of convergence speed, communication rounds, and learning accuracy, marking a significant advancement over existing methodologies.

A. literature review

The nature of the staleness has been one of the core points to drive the convergence guarantees of ASGD. In the literature, several studies analyzed the ASGD, considering different structure for the staleness. In [8], and [21] the authors assumed the staleness to be fixed among all workers, while in [10], [13], [22], the staleness is assumed to be changing but upper bounded. In [23] the delay is assumed to be random with bounded expectation and in [24] the delay is assumed to be growing Polynomially. The following approaches have been proposed to deal with the delayed gradients:

- **Adaptive Learning Rate Methods:** [8], [17], [18], [25], [26] dynamically adjust the learning rate based on gradient staleness (or statically [3], [27] based on the staleness upper bound) aiming to counteract the negative effects of delayed gradients. These methods attempt to balance the step size in gradient descent to ensure stable and effective convergence. However, this approach can be resource-intensive and may result in excessive gradient discounting, potentially hindering the learning process.
- **Gradient Manipulation Techniques:** In this line of research, the delayed gradients are processed further to mitigate the effect of the delay. For example, in [7], the authors aim to correct stale gradients by adjusting them based on the delay, often requiring the computation of additional parameters like gradient norms and even approximations of the Hessian matrix. While in [28], the authors handled the delayed gradients by adopting SGD's acceleration techniques, such as variance reduction, Stochastic Coordinate Sampling, and Nesterov's Acceleration techniques. Despite their novelty, these methods are often limited to scenarios where the objective function and its derivatives exhibit certain smoothness properties, which may not always be the case in practical applications.

Although these studies introduce a stable version of ASGD, the performance did not improve [20]. Specifically, the convergence of most of these approaches is mainly governed by the staleness bound or average as shown in table I. Also, many of their experiments use few workers, which may not reflect the algorithms' scalability and efficiency in larger distributed systems [3], [21].

On the other hand, synchronous SGD allows all workers to contribute to the update of the global model simultaneously by synchronizing the gradient computations across all workers at each iteration. This synchronization step guarantees that the model parameters are updated using gradients computed from the same version of the model, maintaining uniformity in the learning process.

While synchronous SGD has been widely used in training large-scale deep neural networks, it leads to bottlenecks due to the slow workers(stragglers) [29]. Also, it requires very high communication bandwidth between computational workers to exchange gradients and parameters between all workers iteratively. The communication overhead can become a limiting factor, as the ratio of communication time to training time per epoch increases linearly with the number of workers [4], leading to scalability issues beyond a certain number of nodes. This is particularly relevant in the context of emerging 6G networks, which necessitate highly efficient communication protocols to handle the increased demands of distributed learning applications [30], [31].

For generally non-convex smooth functions the convergence is goverend by the term $\frac{1}{\sqrt{T}}$. In Table I we compare the relevant approaches that addressed the delayed gradient problem, both in distributed and federated learning with our proposed algorithms. As shown, we are able to provide a staleness free convergence rate without sophisticated restrictions on the learning rate or the function properties.

II. Problem Setup

In many machine learning and optimization tasks, we aim to find the optimal set of parameters x^* in $\mathbb{R}^d$ that minimize a loss function [19], which can be formally stated as:

$$\min_{x \in \mathbb{R}^d} F(x) := \mathbb{E}_{\xi \sim D}[f(x, \xi)]. \tag{1}$$

Here, $f(x, \xi)$ denotes the loss function evaluated at parameters x with respect to a random data point ξ, which is independently sampled from the data distribution D. The global loss function $F(x)$ represents the expected loss over the data distribution. Various loss functions are employed depending on the specific task. For regression tasks, common loss functions include the Mean Squared Error (MSE), defined as $f(x, \xi) = (y - \hat{y}(x))^2$, and the Mean Absolute Error (MAE), given by $f(x, \xi) = |y - \hat{y}(x)|$, where $y \subset \xi$ is the actual value (ground truth) of the target variable, and $\hat{y}(x)$ is the predicted value of the target variable using paramter x. In multi-class classification settings, the Categorical Cross-Entropy Loss, $f(x, \xi) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c(x))$, is often employed, where $y_c \subset \xi$ is the actual binary label for class c, where $y_c \in \{0, 1\}$ and only one class has the label 1 (one-hot encoding), and $\hat{y}_c(x)$ is the predicted probability that the target variable belongs to class c computed at paramter x.

To solve this optimization problem efficiently, especially when dealing with large-scale data, we use a distributed learning system where multiple clients collaboratively optimize a global loss function under a central server's coordination. The system consists of:

- **Server**: A central entity coordinating training by aggregating gradients and updating global parameters.
- **Clients**: I parallel workers that compute gradients using independently sampled IID batches from the dataset.

In this distributed setup:

1) Each client i independently samples an IID batch ξ_i, with size B, from the entire dataset.
2) Based on the sampled batch ξ_i, each client i computes the average stochastic gradient $g(x_i, \xi_i)$. Here, x_i represents the version of the global parameters that the client i has, which might not be the most recent one at the server due to the asynchronous nature of communication.
3) The server collects these gradients from the clients and aggregates them to update the global parameters.

For simplicity of notation, we drop the batch argument from the gradient symbol and use $g(x^{\tau_i})$ to represent the stochastic gradient computed by client i using parameter x^{τ_i}. x^{τ_i} is the version of the global model parameters available to client i at its local iteration τ_i, corresponding to a certain global iteration n.

Algorithm	Convergence	Staleness	Learning/Weighting Rate	Objective Function Properties
Standard ASGD [11]	$O\left(\frac{\tau}{T}+\frac{\sigma^2}{\sqrt{T}}\right)$	Upper-bounded	Staleness-dependent	L-Smooth Non-convex
Standard Synchronous SGD [15]	$O\left(\frac{I}{T}+\frac{\sigma^2}{\sqrt{T}}\right)$	Zero staleness via full synchrony	Staleness-independent	L-Smooth Non-convex
Delay Compensation ASGD [7]	$O\left(\frac{\sigma^*}{\sqrt{T}}\right)$	Fixed	Staleness-independent	Smooth, L-Lipschitz activation, μ-strongly convex locally, bounded first, second and third derivatives
Coherence-based ASGD [3]	$O\left(\frac{\tau}{\sqrt{T}}+\frac{\sigma\log T}{\tau\sqrt{T}}\right)$	Upper-bounded	Staleness-dependent	L-smooth and μ-weakly convex
Staleness adaptive ASGD [20][*]	$O\left(\frac{I}{T}+\frac{\sigma}{\sqrt{T}}\right)$	Not relying on any staleness assumption	Staleness-dependent	L-Smooth Non-convex
s-QASY (Proposed)	$O\left(\frac{\sigma^2}{\sqrt{T}}\right)$	Not relying on any staleness assumption	Staleness-independent	L-Smooth Non-convex
QASY (Proposed)	$O\left(\frac{\sigma^2}{\sqrt{T}}\right)$	Not relying on any staleness assumption	Staleness-independent	L-Smooth Non-convex

[*] Although this bound appears to be independent of staleness, it is inherently related to the number of workers. Increasing the number of workers leads to greater staleness, implying that staleness is still implicitly present. Furthermore, utilizing all stale gradients with a discounted learning rate results in excessive communication overhead without a corresponding improvement in learning performance, as demonstrated in the simulations.

TABLE I: Comparison of different asynchronous optimization approaches, where T is the number of iterations, I is the number of workers, τ is the staleness upper bound, σ^* is the variance of the delay compensated gradient, and L denotes the Lipschitz and smoothness parameters of the objective function.

After the central server receives a gradient from client i and decides to update the global model parameters x^n at iteration n, the update is performed as:

$$x^{n+1} = x^n - \eta g(x^{\tau_i}).$$

where η denotes the global learning rate. The optimization process thus involves clients and the server working collaboratively to minimize the global loss function $F(x)$. This distributed approach enhances computational efficiency and scalability, allowing the system to handle large datasets and complex models more effectively. The asynchronous nature of the system ensures that clients do not have to wait for the most recent global parameters, further improving the system's efficiency. Conversely, Fully Synchronous SGD (FSSGD) has been shown to consistently achieve higher accuracy due to its use of up-to-date gradients, aligning closely with the current state of the global model. However, this method comes at the cost of increased communication rounds for synchronization, necessitating more extensive resource utilization.

While using delayed gradients relieves the stress of the communication overhead required by synchronous approaches, the asynchronous methods also cause the resources to inflate due to the instability presented by the delayed gradient. As a result, the need for a framework that estimates the quality of the gradients before using them to evaluate the model is crucial. Since the algorithms main goal is to improve the leaning performance while maintaining a reasonable resources consumption, along with the *convergence rate*, we use the *cumulative time*, which is the training time measured in time units (seconds, minutes,...etc) and the number of *communication rounds* between the workers and the server as performance metrics to evaluate our algorithms. One communication round is defined as the server communicating a parameter to the worker or the worker communicating a gradient to the server.[1]

III. Gradient Quality-Aware Asynchronous Approach

In this section, we introduce a Quality-based selection approach for delayed gradients, designed to enhance the communication efficiency and reliability of gradient processing in an asynchronous setting. The delayed gradients sent by the workers to the server may be helpful to minimize the objective function. On the other hand, some of the gradients can hinder the convergence leading to adversary like behavior.

Intuition: To illustrate our intuition, consider minimizing the function $f(x) = x^2$ using gradient descent starting from a point $x^n = 5$, the traditional gradient descent update with a learning rate $\eta = 0.1$ results in $x^{n+1} = 4$. Alternatively, we can use a gradient computed at another point $y = 20$. In this case the new point will be $x^{n+1} = 5 - 0.1 \times 2 \times 20 = 1$, which is closer to the optimal value than the point derived from using x^n. However, if we compute the gradient at point $z = -10$, it results in $x^{n+1} = 5 - 0.1 \cdot 2 \cdot (-10) = 7$ which is farther from the optimal point than x^t. This example supports the insight that not all delayed gradients are bad; indeed, under certain circumstances, they can be harnessed to expedite the convergence towards the minimum. Note that, the distance between x^n and y is the same as the distance between x^n and z, indicating that the distance between the points is not the only factor that decides the usefulness of the delayed gradient as considered in the earlier works [20]. Indeed, the norm of

[1]Since gradients and parameters are vectors on the order of millions, sending a single real number, such as the gradient norm, is not considered a communication round.

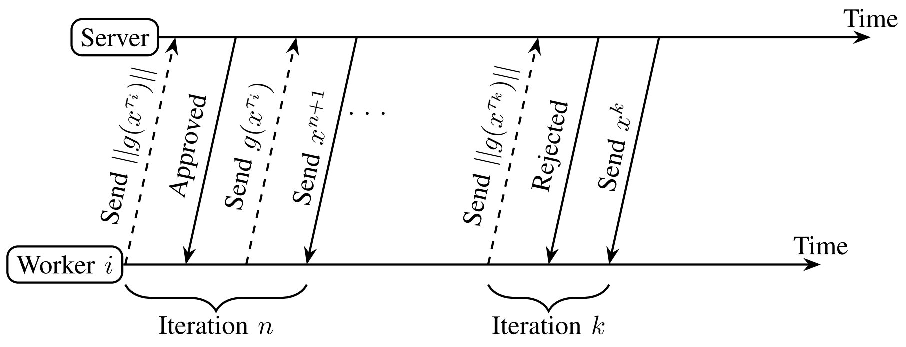

Fig. 1: Time Sequence Diagram illustrating the chronological interactions between the server and Worker i. The process flow covers two iterations: In iteration n, Worker i sends an accepted gradient to the server, followed by iteration k where the gradient sent by Worker i is rejected. This exemplifies the decision-making process of the Quality based selection of the gradients asynchronously.

the delayed gradient, smoothness, and the learning rate are also significant factors. For example, the norm of the gradient at point $y = 20$ is much higher compared to that of at point $z = -10$.

Key component of our algorithm: Using the above intuition, we are now ready to discuss the key novel aspect of our proposed algorithm compare to the existing methods. We use the ratio between the norm of the gradient $g(x^{\tau_i})$ and the distance between the stale model parameter at and the current global model $||x^{\tau_i} - x^n||_2$ as a factor to determine whether to accept the stale gradient or not. In particular, if the ratio is above a threshold θ we accept the gradient otherwise we reject it.

The above serves two purposes– i) The worker does not need to send the computed gradient immediately saving a lot of communication resource; rather, the agent only needs to send the norm of the gradient. ii) By carefully choosing the threshold θ (Section IV), we can achieve the state-of-the-art convergence rate $O(1/\sqrt{T})$ devoid of any staleness parameter. Hence, we can achieve the same convergence rate as that of the synchronous SGD while using less communication resources. Note that unlike [8], [20] we do not accept all the gradients and then scale down the learning rate based on the stale value. *Rather, we choose to accept those gradients that can provably contribute towards convergence without scaling down the learning rate*. Hence, we achieve similar convergence rate without *using large communication resources*.

Algorithm: The QASY algorithm (Algorithm 1) starts by initializing the model parameters and setting up a mechanism to store gradients persistently. It operates in iterations, each beginning with receiving a notification from any worker i_n that a gradient has been computed, accompanied by the norm of that gradient (line 8). The server then decides whether to accept the gradient for updating the model or to reject it, thereby saving a communication round. The handling of the gradient proceeds as follows:

1) **Immediate Acceptance for Recent Gradients:** If the received gradient $g(x^{\tau_{i_n}})$ is recent ($\tau_{i_n} = n$), it is immediately accepted without further evaluation (line 9).
2) **Quality Check for Delayed Gradients:** The norm of the delayed gradient $g(x^{\tau_{i_n}})$ with $\tau_{i_n} < n$ is assessed for quality. The algorithm checks if $\frac{||g(x^{\tau_{i_n}})||}{||x_n - x^{\tau_{i_n}}||} \geq \theta$. If the condition is satisfied, the worker sends the gradient as it is accepted and used for the model update (line 10). Otherwise, the worker is instructed to recompute the gradient based on the latest model parameters x^n sent by the server (lines 22-24).

The update process in QASY begins once the server receives an accepted gradient. The server updates the model into a temporary variable x^{temp} (line 12) and proceeds to the refinement step. In this refinement step, the server reviews gradients stored from previous iterations to identify any that are still relevant and impactful for updating the current model state (lines 13-17). This involves checking if a stored stochastic gradient $g(x^{\tau_i})$, when applied to x^{temp}, would still satisfy the quality criterion based on the norm and threshold θ. By updating the model with these refined gradients, the learning process becomes more effective and precise. This allows workers who contributed quality gradients to proceed with computations reflecting enhanced collective insights, rather than just the most recent updates. Additionally, reusing gradients conserves resources since we do not need to compute any gradient.

After completing this refinement, the server finalizes the model update, resulting in the new model x^{n+1} (line 18). This updated model is then sent *only* to the worker that provided the accepted gradient (line 19), as detailed in Algorithm 1, hence, unlike the *synchronized version*, the algorithm does not send the updated model to every worker.

Despite the algorithm's effectiveness in handling gradients and improving learning convergence, QASY has a higher storage and computational cost cost as it needs to store earlier gradients and use those for model update. We also propose a simpler variant, simplified-Quality Aware Asynchronous SGD (s-QASY) that does not need to store earlier gradients. In other words, s-QASY skips the refinement step in Algorithm 1 (lines 13-17), which reduces computational complexity. We

Algorithm 1 Quality Aware Asynchronous SGD (QAYS)

```
Require: learning rate $\eta$, number of workers $I$, number of iterations $T$, acceptance threshold $\theta$
1:  Initialize $x^0 \in \mathbb{R}^d$.
2:  Set $\tau_i = 0$ for all $i = 1, \ldots, I$.
3:  Initialize set $S \leftarrow \emptyset$, $x^{temp}$.
4:  for $n = 0, 1, 2, \ldots, T-1$ do
5:      Set flag $flag \leftarrow$ True
6:      $x^{temp} = x^n$
7:      while flag do
8:          Wait for the next worker to finish computing the gradient, and denote this worker as $i_n$.
9:          Worker $i_n$ sends $\|g(x^{\tau_{i_n}})\|$.
10:         if $\tau_{i_n} = n$ OR $\frac{\|g(x^{\tau_{i_n}})\|}{\|x^n - x^{\tau_{i_n}}\|} \geq \theta$ then
11:             Worker $i_n$ sends $g(x^{\tau_{i_n}})$.
12:             $S \leftarrow S \cup \{(i_n, g(x^{\tau_{i_n}}))\}$
13:             $x^{temp} = x^{temp} - \eta\, g(x^{\tau_{i_n}})$
14:             for each $(i, g(x^{\tau_i})) \in S$ do
15:                 if $\frac{\|g_i(x^{\tau_i})\|}{\|x^n - x^{\tau_i}\|} \geq \theta$ then
16:                     $x^{temp} = x^{temp} - \eta\, g(x^{\tau_{i_n}})$
17:                 end if
18:             end for
19:             $x^{n+1} = x^{temp}$
20:             Store $x^{n+1}$.
21:             Send $x^{n+1}$ to worker $i_n$.
22:             $\tau_{i_n} \leftarrow n + 1$
23:             $flag \leftarrow$ False
24:         else
25:             Send $x^n$ to worker $i_n$.
26:         end if
27:     end while
28: end for
```

show that both the versions have a similar convergence rate though QASY performs *slightly* better empirically.

IV. Convergence Analysis

We first prove the convergence of the s-QASY and subsequently, we prove the convergence rate of QASY.

We begin by stating the assumptions required for the convergence analysis:

- **Assumption 1 (Boundedness and Smoothness):** F is bounded and L-smooth, i.e., for all $u, v \in \mathbb{R}^d$,
$$\|\nabla F(u) - \nabla F(v)\| \leq L\|u - v\|.$$
- **Assumption 2 (Unbiasedness and Bounded Variance):** For any $x \in \mathbb{R}^d$, the gradient estimator $g(x)$ satisfies:
  - **Unbiasedness:** $\mathbb{E}[g(x)|x] = \nabla F(x)$.
  - **Bounded Variance:** $\mathbb{E}[\|g(x) - \nabla F(x)\|^2|x] \leq \frac{\sigma^2}{B}$,

  where B is the batch size.

Our assumptions align with those commonly adopted in the literature on distributed learning frameworks [3], [7], [10], [11]. Specifically, we do not require the bounded staleness assumption, which further emphasizes the robustness and generality of our approach. Now, we will establish an upper bound on the one iteration cost reduction when multiple delayed gradients are used every iteration. Specifically, in the next lemma, at any iteration n, we will consider applying K number of gradients stored at the set $\mathcal{U}_n$. This upper bound is general and applies to both s-QASY and QASY algorithms. For the s-QASY algorithm, $K = 1$, indicating that only one delayed gradient is applied per iteration. For the QASY algorithm, K can vary at each iteration, allowing the application of multiple delayed gradients.

Lemma 1. *For the sequence $\{x^n\}$ generated by the following update rule:*
$$x^{n+1} = x^n - \eta \sum_{g(x^i)\in\mathcal{U}_n} g(x^i), \tag{2}$$
where $\mathcal{U}_n$ is the set of K gradients, the one iteration cost reduction can be upper bounded as follows:
$$F(x^{n+1}) - F(x^n) \leq -\frac{\eta}{2}K||\nabla F(x^n)||^2 + \frac{\eta}{2}\sum_{g(x^i)\in\mathcal{U}_n} \big(2L^2||x^n - x^i||^2 - (1 - LK\eta)||g(x^i)||^2 + 2||\nabla F(x^i) - g(x^i)||^2\big)\,. \tag{3}$$

The proof of this lemma can be found in Appendix VII-A.

Note that Equation (3), lists out the main components that determine the effectiveness of the delayed gradient. While the first and last terms are presumably unknown and can not be controlled by the server's communication decisions, the two middle terms are known by the server and can be fully utilized to improve the reduction. In the next theorems, we will design a threshold based on which we can ensure that the applied delayed gradient is as useful as the recent SGD vector.

Theorem 1. *Given the sequence $\{x^n\}$ generated by s-QASY Algorithm, and under the previously stated assumptions and the threshold parameter $\theta = \frac{\sqrt{2}L}{\sqrt{1-L\eta}}$, with $B^2 < T$ the following inequality holds for the step size η satisfying $\eta = \frac{B}{L\sqrt{T}}$:*
$$\frac{1}{T}\sum_{n=0}^{T-1} \mathbb{E}\|\nabla F(x^n)\|^2 \leq \frac{2L\mathbb{E}[F(x^0) - F(x^T)]}{B\sqrt{T}} + 2\frac{\sigma^2}{L\sqrt{T}}. \tag{4}$$

The proof of this Theorem can be found in Appendix VII-B.

Next, we will consider the implications of this result for the QASY algorithm, where the number of updates every iterations is not fixed and random.

Theorem 2. *Given the sequence $\{x^n\}$ generated by QASY algorithm with $\theta = \frac{\sqrt{2}L}{\sqrt{1-ML\eta}}$, $\eta = \frac{B}{ML\sqrt{T}}$, and $T > B^2$, where M is the maximum number of updates in any iteration, then the following inequality hold:*
$$\frac{1}{T}\sum_{n=0}^{T-1} \mathbb{E}[\|\nabla F(x^n)\|^2] \leq \frac{2ML(\mathbb{E}[F(x^0)] - \mathbb{E}[F(x^T)])}{B\sqrt{T}} + \frac{2\sigma^2}{L\sqrt{T}}. \tag{5}$$

The proof of this Theorem can be found in Appendix VII-C.

Remark 1. *Both QASY, and s-QASY's convergence rate and step size, do not depend on the gradients' staleness. In particular, we obtain the same convergence rate* $O(1/\sqrt{T})$ *as the synchronous SGD using asynchronous method.*

[20] achieved a similar convergence rate without the stale parameter, however, they used all the gradients. Thus, they require a lot of communication rounds as we observe in the empirical results. Even though theoretical bound in [20] is similar to that of ours, we outperform them in the empirical evaluations.

We observe that as T increases, the threshold decreases. Intuitively, higher T indicates that we can recover from accepting wrong stale gradients, hence, we can accept more stale gradients. Hence, as T increases, the algorithm can be more aggressive. However, the maximum value of θ is L. Hence, even when $T \to \infty$, it does not accept all the stale gradients unlike the existing approaches for asynchronous SGD.

Note that though the learning rate (and thus, the threshold) depends on the information of B and L, in the empirical results, we do not rely on those information.

V. Simulation Results

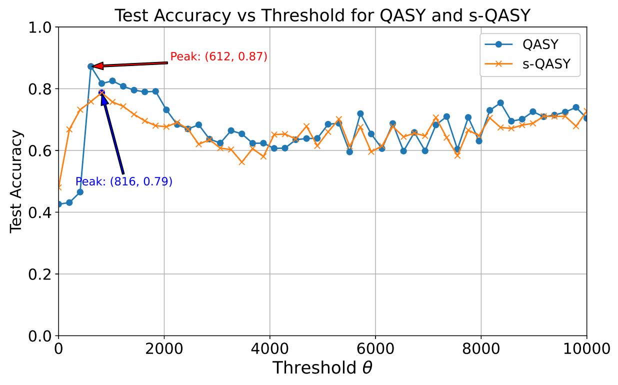

Fig. 2: Relationship between Test Accuracy and Threshold value for QASY and s-QASY.

In this section we present the results of our experiments that used to assess the performance of our algorithms QASY and s-QASY. We consider a fully connected feedforward neural network with the following structure:

1) **Input Layer:** 784 neurons, corresponding to the 28x28 pixel input images.
2) **Hidden Layers:**
   - **First Hidden Layer:** 128 neurons, ReLU activation.
   - **Second Hidden Layer:** 64 neurons, ReLU activation.
3) **Output Layer:** 10 neurons, softmax activation for multi-class classification.

This model, implemented using TensorFlow Keras, is designed for image classification tasks, leveraging ReLU activation for non-linearity and computational efficiency, and

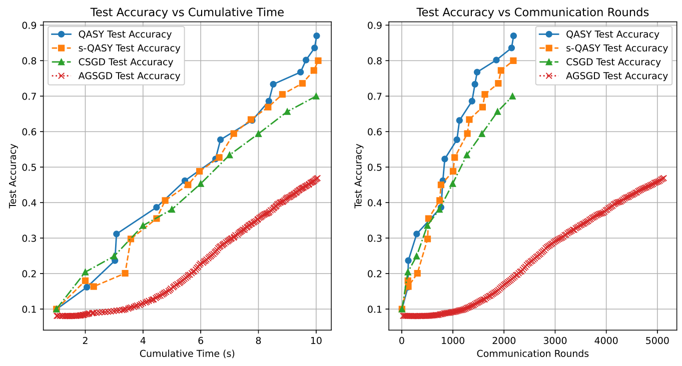

Fig. 3: Test accuracy comparison of AGSGD, QASY, s-QASY, and CSGD with a batch size of 100 vs cumulative time and communication rounds

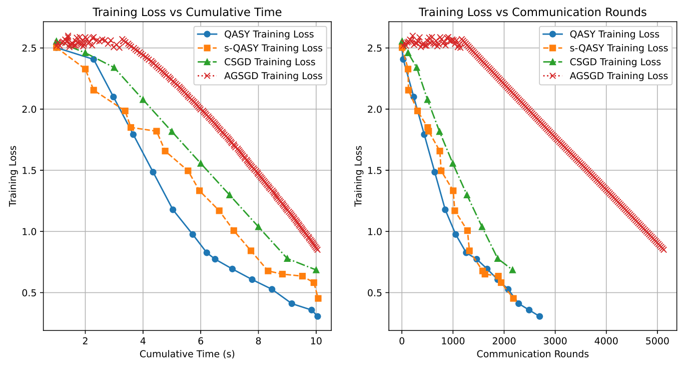

Fig. 4: Training loss comparison of AGSGD, QASY, s-QASY, and CSGD with a batch size of 100 vs cumulative time and communication rounds

softmax for outputting class probabilities. The MNIST dataset [32] was used in our experiments with the categorical cross entropy loss.

Baseline Algorithms: As the severity of the staleness appears in the larger scale setup, we evaluated the performance across $I = 1000$ workers, with fixed computational delays evenly spaced from 1 to 10 seconds. However, it would be unfair to compare FSSGD [15] (because of excessive communication rounds) with our schemes. Therefore, we introduce a more practical synchronous version **Conservative Stochastic Gradient Descent (CSGD)**, which is the same as s-QASY except that $\theta = \infty$. Thus, CSGD is a communication efficient version of the synchronous stochastic gradient descent, that only accepts the recent gradient and discards any delayed one (assuming $1/0 \geq \infty$). We also consider when $\theta = 0$ in s-QASY, where the server aggressively updates the model with any received gradient, so we call it **Aggressive Stochastic Gradient Descent (AGSGD)**. In other words, ASGD is the traditional asynchronous stochastic gradient descent algorithm that accepts any stale gradient.

Learning Rate: The learning rate is set to be 0.001 for CSGD, and s-QASY, which is the best practical value based on our preliminary experiments, while QASY's learning rate is chosen to be 10^{-4}. For ASGD, we used a staleness-scaled learning rate, as proposed in [20] and [8], where the learning rate for a gradient delayed by k iterations is scaled by a factor of $\frac{0.001}{k}$ as it shows the most stable behaviour among other tested approaches. Also, we use a batch size of 100.

Threshold: The performance of QASY and s-QASY algorithms are mainly determined by the choice of the threshold θ. A lower threshold, similar to the behavior of the AGSGD algorithm, allows for the acceptance of all gradients regardless of their staleness, while a higher threshold mirrors the CSGD algorithm's strategy of considering only the most recent gradients. As the threshold is essentially a function of the hyper-parameters (learning rate and smoothness), it can be optimized empirically using standard tuning techniques [33]–[35]. For instance, bi-level optimization [33] iterates between updating the threshold in the outer loop and solving the training task in the inner loop. Alternatively, Bayesian optimization [34] constructs a surrogate model (e.g., Gaussian Process) to guide the search toward promising hyper-parameter configurations based on observed performance. Methods like random or grid search [35] can also be effective in lower-dimensional settings or under tighter computational budgets.

Figure 2 illustrates the empirical relationship between the threshold (θ) and the test accuracy of the QASY and s-QASY algorithms. Notably, both QASY and s-QASY demonstrate marked improvements in test accuracy when the threshold increases from zero, suggesting that even a modest degree of filtering on delayed gradients can significantly stabilize the training process. Optimal accuracy is achieved within a threshold window estimated to span from 600 to 800, highlighting an optimal trade-off where the algorithms benefit from a prudent mix of gradient exploitation and staleness mitigation. This confluence of precision in gradient utilization underpins the peak learning performance for these algorithms. Moreover, the performance of QASY and s-QASY closely aligns across various thresholds, indicating that both algorithms leverage gradient staleness to a similar extent and efficacy. However, as the threshold rate increases beyond the optimal range, a gradual decline in test accuracy is observed, followed by a plateau. This phenomenon suggests that there exists a threshold saturation point beyond which the accepting of stale gradients in fact contribute to reduce the accuracy gains, with performance asymptotically approaching that of the CSGD.

Performance: In Figures 3 and 4, we compare our proposed QASY and s-QASY with two baseline algorithms in terms of the test accuracy and training loss against the actual training time and the number of communication rounds. To ensure a fair comparison, the training duration for all four algorithms is fixed at 10 seconds. This fixed training time allows us to evaluate the efficiency of each algorithm based on the number of communication rounds consumed. We use the best threshold for QASY and s-QASY as we obtained in Figure 2. As shown in Figures, both QASY, and s-QASY outperform AGSGD and CSGD in terms of the training loss and test accuracy. Remarkably, our proposed algorithms reached a test accuracy of 87% and 79%, respectively. which out performs its synchronous counterpart, the CSGD, that only reach 70%. Also note that the ASGD has the worst performance, as the high staleness causes that most of the delayed gradient to lose their potential due to the excessive down-scaling of the learning rate. In addition to its poor performance, ASGD consumes almost 2 times the communication rounds that other algorithms use in their training. The performance of QASY is better compared to s-QASY however it comes at the cost of a slightly higher computational cost as we discussed.

VI. Conclusion and Future Work

In this paper, we introduced QASY, a new method to make learning in a distributed setting—where many computers work together—more effective, especially when updates happen at different times. We developed QASY to smartly choose which updates to use, focusing on their quality. We also created a simpler version, s-QASY, to make the process easier and less demanding.

Additionally, this work opens the door to further exploration into how our methods can benefit the network as a whole, by possibly reducing communication needs and enhancing overall efficiency. Future research could extend our approach to the federated learning setting, where data and resources vary widely across different locations. Applying our algorithms in such environments could offer valuable insights into their adaptability and effectiveness in handling data and resource heterogeneity, potentially leading to more robust and scalable learning solutions. Moreover, future work could extend the convergence analysis to encompass the strongly convex case, providing a more comprehensive understanding of our algorithms' performance under diverse mathematical conditions.

VII. Appendix

A. *proof of Lemma 1*

Proof. By considering the update rule in (2), and using the L-smoothness of F, we have for any n:

$$F(x^{n+1}) - F(x^n) \leq \langle \nabla F(x^n), x^{n+1} - x^n \rangle + \frac{L}{2}\|x^{n+1} - x^n\|^2 \tag{6}$$

$$= \langle \nabla F(x^n), -\eta \sum_{g(x^i)\in\mathcal{U}_n} g(x^{\tau_{i_n}}) \rangle + \frac{L\eta^2}{2}\left\| \sum_{g(x^i)\in\mathcal{U}_n} g(x^i) \right\|^2 . \tag{7}$$

Decomposing the inner product:

$$\langle \nabla F(x^n), \sum_{g(x^i)\in\mathcal{U}_n} g(x^i) \rangle = \sum_{g(x^i)\in\mathcal{U}_n} \langle \nabla F(x^n), g(x^i) \rangle.$$

Using the identity:

$$\langle a, b \rangle = \frac{1}{2}(\|a\|^2 + \|b\|^2 - \|a-b\|^2),$$

we get:

$$\langle \nabla F(x^n), g(x^i) \rangle = \frac{1}{2}\left(\|\nabla F(x^n)\|^2 + \|g(x^i)\|^2 - \|\nabla F(x^n) - g(x^i)\|^2\right). \tag{8}$$

Substituting back into the inequality:

$$F(x^{n+1}) - F(x^n) \leq -\eta \Bigg(\frac{1}{2} K \|\nabla F(x^n)\|^2 + \frac{1}{2} \sum_{g(x^i)\in\mathcal{U}_n} \|g(x^i)\|^2 - \frac{1}{2} \sum_{i_n\in\mathcal{U}_n} \|\nabla F(x^n) - g(x^{\tau_{i_n}})\|^2 \Bigg) + \frac{L\eta^2}{2} \left\| \sum_{g(x^i)\in\mathcal{U}_n} g(x^i) \right\|^2 . \tag{9}$$

Bounding the squared norm of the sum of the gradients:

$$\left\| \sum_{g(x^i)\in\mathcal{U}_n} g(x^i) \right\|^2 \leq K \sum_{g(x^i)\in\mathcal{U}_n} \|g(x^i)\|^2.$$

Using the inequality $(a+b)^2 \leq 2a^2 + 2b^2$:

$$\sum_{g(x^i)\in\mathcal{U}_n} \|\nabla F(x^n) - g(x^i)\|^2 \leq 2 \sum_{g(x^i)\in\mathcal{U}_n} \|\nabla F(x^n) - \nabla F(x^i)\|^2 + 2 \sum_{g(x^i)\in\mathcal{U}_n} \|\nabla F(x^i) - g(x^i)\|^2. \tag{10}$$

Using L-smoothness:

$$\sum_{g(x^i)\in\mathcal{U}_n} \|\nabla F(x^n) - \nabla F(x^i)\|^2 \leq \sum_{g(x^i)\in\mathcal{U}_n} L^2 \|x^n - x^i\|^2.$$

Substituting the bound into the inequality we get (3). □

B. *Proof of Theorem 1*

Proof. We can get the one iteration cost reduction of s-QASY by substituting into equation 3 with $K = 1$, we get:

$$F(x^{n+1}) - F(x^n) \leq -\frac{\eta}{2} \|\nabla F(x^n)\|^2 + \eta \|\nabla F(x^{\tau_{i_n}}) - g(x^{\tau_{i_n}})\|^2 \quad + \eta L^2 \|x^n - x^{\tau_{i_n}}\|^2 - \frac{\eta}{2}(1 - \eta L) \|g(x^{\tau_{i_n}})\|^2.$$

Since the accepted gradient $g(x^{\tau_{i_n}})$ achieves the condition:

$$\frac{\|g(x^{\tau_{i_n}})\|}{\|x^n - x^{\tau_{i_n}}\|} \geq \theta = \frac{\sqrt{2}L}{\sqrt{1 - L\eta}},$$

we can ensure that:
$\eta L^2 \|x^n - x^{\tau_{i_n}}\|^2 - \frac{\eta}{2}(1 - \eta L) \|g(x^{\tau_{i_n}})\|^2 \leq 0$. Therefore, we can upper bound the expression by removing these two terms:

$$F(x^{n+1}) - F(x^n) \leq -\frac{\eta}{2} \|\nabla F(x^n)\|^2 + \eta \|\nabla F(x^{\tau_{i_n}}) - g(x^{\tau_{i_n}})\|^2.$$

Taking the expectation given i_n, $x^{\tau_{i_n}}$ and x^n, we have:

$$\mathbb{E}[F(x^{n+1}) - F(x^n) \mid i_n, x^n] \leq -\frac{\eta}{2} \|\nabla F(x^n)\|^2 + \eta \mathbb{E} \|\nabla F(x^{\tau_{i_n}}) - g(x^{\tau_{i_n}})\|^2 \leq -\frac{\eta}{2} \|\nabla F(x^n)\|^2 + \eta \frac{\sigma^2}{B}.$$

By taking the full expectation, we get:

$$\mathbb{E}[F(x^{n+1}) - F(x^n)] \leq -\frac{\eta}{2} \mathbb{E} \|\nabla F(x^n)\|^2 + \eta \frac{\sigma^2}{B}.$$

Summing from $n = 0$ to $T - 1$ and dividing by T, we obtain:

$$\frac{\mathbb{E}[F(x^T) - F(x^0)]}{T} \leq -\frac{\eta}{2} \frac{1}{T} \sum_{n=0}^{T-1} \mathbb{E} \|\nabla F(x^n)\|^2 + \eta \frac{\sigma^2}{B}.$$

Rearranging the terms, and substituting with η value, we get the desired inequality:

$$\frac{1}{T} \sum_{n=0}^{T-1} \mathbb{E} \|\nabla F(x^n)\|^2 \leq \frac{2L\mathbb{E}[F(x^0) - F(x^T)]}{B\sqrt{T}} + 2 \frac{\sigma^2}{L\sqrt{T}}.$$

□

C. *Proof of Theorem 2*

Proof. the QASY algorithm update rule can be written as:

$$x^{n+1} = x^n - \eta \sum_{g(x^i)\in\mathcal{U}_n} g(x^i),$$

where $\mathcal{U}_n$ be the set of gradients used to update the model at iteration n, with each gradient satisfying the quality condition:

$$\frac{\|g(x^i)\|}{\|x^n - x^i\|} \geq \theta,$$

So, using equation (3), we can write the one iteration cost reduction as:

$$F(x^{n+1}) - F(x^n) \leq -\frac{\eta}{2} K_n ||\nabla F(x^n)||^2 + \frac{\eta}{2} \sum_{g(x^i)\in\mathcal{U}_n} \big(2L^2 ||x^n - x^i||^2 - (1 - LK_n\eta) ||g(x^i)||^2 + 2||\nabla F(x^i) - g(x^i)||^2 \big). \tag{11}$$

where $K_n = |\mathcal{U}_n|$. Since

$$\frac{\|g(x^i)\|}{\|x^n - x^i\|} \geq \frac{\sqrt{2}L}{\sqrt{1 - ML\eta}} \qquad \forall i \in \mathcal{U}_n,$$

therefore,

$$\frac{\|g(x^i)\|}{\|x^n - x^i\|} \geq \frac{\sqrt{2}L}{\sqrt{1 - K_n L\eta}} \qquad \forall i \in \mathcal{U}_n.$$

Taking the expectation, given $\mathcal{U}_n$, and $\{x^0, ...x^n\}$, we get:

$$F(x^{n+1}) - F(x^n) \leq -\frac{\eta K_n}{2} \|\nabla F(x^n)\|^2 + \eta \mathbb{E} \left[\sum_{g(x^i)\in\mathcal{U}_n} \mathbb{E} \Big[||\nabla F(x^i) - g(x^i)||^2 \mid x^i \Big] \mid \mathcal{U}_n \right] \leq -\frac{\eta K_n}{2} \|\nabla F(x^n)\|^2 + \eta K_n \frac{\sigma^2}{B} \leq -\frac{1}{2} \|\nabla F(x^n)\|^2 + \eta M \frac{\sigma^2}{B}$$

By taking full Expectation we get:

$$\mathbb{E}[F(x^{n+1}) - F(x^n)] \leq -\frac{\eta}{2} \mathbb{E}[\|\nabla F(x^n)\|^2] + \eta M \frac{\sigma^2}{B}.$$

With similar steps to Theorem 1's proof, we can get equation (5). □

References

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

[2] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[3] W. Dai, Y. Zhou, N. Dong, H. Zhang, and E. P. Xing, "Toward understanding the impact of staleness in distributed machine learning," *arXiv preprint arXiv:1810.03264*, 2018.

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

[5] Z. Huo and H. Huang, "Asynchronous stochastic gradient descent with variance reduction for non-convex optimization," *arXiv preprint arXiv:1604.03584*, 2016.

[6] R. Islamov, M. Safaryan, and D. Alistarh, "Asgrad: A sharp unified analysis of asynchronous-sgd algorithms," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2024, pp. 649–657.

[7] H. R. Feyzmahdavian, A. Aytekin, and M. Johansson, "Asynchronous stochastic gradient descent with delay compensation," in *Proceedings of the 33rd International Conference on Machine Learning*, vol. 48, 2016, pp. 2910–2919.

[8] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-sgd for distributed deep learning," *arXiv preprint arXiv:1511.05950*, 2015.

[9] I. Mitliagkas, C. Caramanis, and P. Jain, "Asynchrony begets momentum, with an application to deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2016, pp. 997–1004.

[10] A. Agarwal and J. Duchi, "Distributed delayed stochastic optimization," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE, 2011, pp. 5451–5458.

[11] S. U. Stich and S. P. Karimireddy, "The error-feedback framework: Better rates for sgd with delayed gradients and compressed communication," *arXiv preprint arXiv:1909.05350*, 2019.

[12] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in neural information processing systems*, vol. 30, 2017.

[13] Y. Arjevani, O. Shamir, and N. Srebro, "A tight convergence analysis for stochastic gradient descent with delayed updates," in *Algorithmic Learning Theory*. PMLR, 2020, pp. 111–132.

[14] M. Assran, A. Aytekin, H. R. Feyzmahdavian, M. Johansson, and M. G. Rabbat, "Advances in asynchronous parallel and distributed optimization," *Proceedings of the IEEE*, vol. 108, no. 11, pp. 2013–2031, 2020.

[15] R. M. Gower, N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin, and P. Richtárik, "Sgd: General analysis and improved rates," in *International conference on machine learning*. PMLR, 2019, pp. 5200–5209.

[16] A. Khaled and P. Richtárik, "Better theory for sgd in the nonconvex world," *arXiv preprint arXiv:2002.03329*, 2020.

[17] Y. Chen, Z. Qin, J. Wang, Y. Yu, Y. Gao, and X. Ma, "Asynchronous federated learning for geospatial applications," *IEEE Geoscience and Remote Sensing Letters*, vol. 17, no. 3, pp. 394–398, 2020.

[18] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 3581–3607.

[19] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[20] K. Mishchenko, F. Bach, M. Even, and B. E. Woodworth, "Asynchronous sgd beats minibatch sgd under arbitrary delays," *Advances in Neural Information Processing Systems*, vol. 35, pp. 420–433, 2022.

[21] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," *Advances in neural information processing systems*, vol. 26, 2013.

[22] H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan, "Perturbed iterate analysis for asynchronous stochastic optimization," *SIAM Journal on Optimization*, vol. 27, no. 4, pp. 2202–2229, 2017. [Online]. Available: https://doi.org/10.1137/16M1057000

[23] S. Sra, A. W. Yu, M. Li, and A. Smola, "Adadelay: Delay adaptive distributed stochastic optimization," in *Artificial Intelligence and Statistics*. PMLR, 2016, pp. 957–965.

[24] Z. Zhou, P. Mertikopoulos, N. Bambos, P. Glynn, Y. Ye, L.-J. Li, and L. Fei-Fei, "Distributed asynchronous optimization with unbounded delays: How slow can you go?" in *International Conference on Machine Learning*. PMLR, 2018, pp. 5970–5979.

[25] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[26] Q. Wang, Q. Yang, S. He, Z. Shi, and J. Chen, "Asyncfeded: Asynchronous federated learning with euclidean distance based adaptive weight aggregation," *arXiv preprint arXiv:2205.13797*, 2022.

[27] J. Langford, A. Smola, and M. Zinkevich, "Slow learners are fast," *arXiv preprint arXiv:0911.0491*, 2009.

[28] Q. Meng, W. Chen, J. Yu, T. Wang, Z. Ma, and T.-Y. Liu, "Asynchronous accelerated stochastic gradient descent." in *IJCAI*, 2016, pp. 1853–1859.

[29] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," in *International Conference on Learning Representations Workshop Track*, 2016.

[30] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.

[31] S. Jere, Y. Song, Y. Yi, and L. Liu, "Distributed learning meets 6g: A communication and computing perspective," *IEEE Wireless Communications*, vol. 30, no. 1, pp. 112–117, 2023.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[33] S. Dempe, *Foundations of bilevel programming*. Springer Science & Business Media, 2002.

[34] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[35] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.