Taming Momentum in a Distributed Asynchronous Environment

Ido Hakimi
Department of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel
ido@hakimi@gmail.com

Saar Barkai
Department of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel
saarbarkai@gmail.com

Moshe Gabel
Department of Computer and Mathematical Sciences
University of Toronto Scarborough
Toronto, Canada
mgabel@cs.toronto.edu

Assaf Schuster
Department of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel
assaf@cs.technion.ac.il

Abstract

Although distributed computing can significantly reduce the training time of deep neural networks, scaling the training process while maintaining high efficiency and final accuracy is challenging. Distributed asynchronous training enjoys near-linear speedup, but asynchrony causes gradient staleness, the main difficulty in scaling stochastic gradient descent to large clusters. Momentum, which is often used to accelerate convergence and escape local minima, exacerbates the gradient staleness, thereby hindering convergence. We propose DANA: a novel asynchronous distributed technique which is based on a new gradient staleness measure that we call the gap. By minimizing the gap, DANA mitigates the gradient staleness, despite using momentum, and therefore scales to large clusters while maintaining high final accuracy and fast convergence. DANA adapts Nesterov’s Accelerated Gradient to a distributed setting, computing the gradient on an estimated future position of the model’s parameters. In turn, we show that DANA’s estimation of the future position amplifies the use of a Taylor expansion, which relies on a fast Hessian approximation, making it much more effective and accurate. Our evaluation on the CIFAR and ImageNet datasets shows that DANA outperforms existing methods, in both final accuracy and convergence speed.

1 Introduction

Modern deep neural networks are comprised of millions of parameters, which require massive amounts of data and long training time [25]. The steady growth of these networks over the years has made it impractical to train them from scratch on a single worker (computational device). Distributing the computations over several workers can drastically reduce the training time [5]. Unfortunately, stochastic gradient descent (SGD), which is typically used to train these networks, is an inherently

Preprint. Under review.
sequential algorithm. Thus, training deep neural networks on multiple workers is difficult, especially when trying to maintain fast convergence rate and high final accuracy.

Computations are commonly distributed using data parallelism: data is split across multiple workers and each worker computes over its own data. Synchronous SGD (SSGD) is a straightforward method to distribute the training process of neural networks: each worker computes the gradient over its own separate mini-batches, which are then aggregated to update a single model. SSGD relies on synchronizations to coordinate the workers; this limits its progress to the slowest worker.

Asynchronous SGD (ASGD) addresses the drawbacks of SSGD by eliminating synchronization between the workers [5], therefore scaling almost linearly. However, eliminating the synchronizations induces gradient staleness: gradients sent by workers are often based on parameters that are older than the master’s (parameter server) current parameters. Gradient staleness is one of the main difficulties in scaling ASGD, since it worsens as the number of workers grows. Consequently distributed ASGD suffers from slow convergence and reduced final accuracy, and may not converge at all if the number of workers is high [34].

Momentum [22] has been demonstrated to accelerate SGD convergence and reduce oscillation [26]. Momentum is crucial for achieving state-of-the-art accuracy [26] and is typically used for training deep neural networks [9, 31]. However, when paired with ASGD, momentum exacerbates the gradient staleness [19, 4], up to a point that it diverges when trained on large clusters.

Our contribution: We propose DANA: a novel asynchronous distributed technique which is based on a new gradient staleness measure that we call the gap. By adapting Nesterov’s Accelerated Gradient to a distributed setting, DANA computes the gradient on an estimated future position of the model’s parameters, thereby mitigating the gap. Thus, DANA efficiently scales to large clusters, despite using momentum, while maintaining high final accuracy and fast convergence. Throughout our evaluations, DANA consistently outperformed other ASGD methods, in both final accuracy and convergence speed.

2 Background

The goal of SGD is to minimize an optimization problem \( J(\theta) \) where \( J \) is the objective function (i.e., loss) and the vector \( \theta \in \mathbb{R}^k \) is the model’s parameters from dimensional \( k \). The value of \( J(\theta) \) gives a measure of how far from perfect the performance of the neural network is. Let \( \nabla J(\theta) \) be the gradient of \( J \) with respect to its argument \( \theta \). Then the iterative update rule of sequential SGD for the given problem with learning rate \( \eta \) is:

\[
g_t = \nabla J(\theta_t), \quad \theta_{t+1} = \theta_t - \eta g_t
\]  

Momentum Momentum [22] has been demonstrated to accelerate SGD convergence and reduce oscillation [26]. Momentum can be compared to a heavy ball rolling downhill that accumulates speed on its way towards the minima. The gathered inertia accelerates and smoothes the descent, which helps dampen oscillations and overcome narrow valleys, small humps and local minima [7]. Mathematically, the momentum update rule, without dampening, is simply an exponentially-weighted moving average of gradients that works by adding a fraction \( \gamma \) of the previous momentum vector \( v_{t-1} \) to the current momentum vector \( v_t \):

\[
v_t = \gamma v_{t-1} + g_t, \quad \theta_{t+1} = \theta_t - \eta v_t
\]  

When successive gradients have similar direction, momentum results in larger update steps (higher speed), yielding up to quadratic speedup in convergence rate for SGD [16, 15].

Nesterov In the analogy of the heavy ball rolling downhill, higher speed might make the heavy ball overshoot the bottom of the valley (the minima), if it does not slow down in time. Nesterov [20] proposed Nesterov’s Accelerated Gradient (NAG), which gives the ball a “sense” of where it is going, allowing it to slow down in advance. NAG approximates \( \hat{\theta}_t \), the future value of \( \theta_t \), based on the previous momentum vector \( v_t \):

\[
\hat{\theta}_t = \theta_t - \eta \gamma v_{t-1}
\]  

NAG computes the gradient on the parameters’ approximated future value \( \hat{\theta} \) instead of their current value \( \theta \). Thus, NAG slows down near the minima instead of overshooting the goal and climbing
Assumption 1. The gradient of $\theta_t$ is stale, since it is computed from parameters $\theta_t$ but applied to $\theta_{t+\tau}$.

Remark 1. The difference between the updated parameters $\theta_{t+1}$ and the approximated future position $\hat{\theta}_t$ is only affected by the newly computed gradient $g_t$, and not by $v_t$. Therefore, NAG can accurately compute the gradient even when the momentum vector $v_t$ is large.

$\theta_{t+1} - \hat{\theta}_t = -\eta g_t$

3 Gradient Staleness and Momentum

Figure 1 illustrates the ASGD training process and the origin of gradient staleness. In ASGD training, each worker pulls up-to-date parameters $\theta_t$ from the master and computes the gradient of a single batch (Algorithm 1). Once the computations finish, the worker sends the gradient $g_t$ back to the master. The master (Algorithm 2) then applies the gradient $g_t$ to its current set of parameters $\theta_{t+\tau}$, where $\tau$ is the lag. The lag is defined as the number of updates the master has received from other workers while worker $i$ was computing $g_t$.

Algorithm 1 ASGD: worker

- Receive parameters $\theta_t$ from master
- Compute gradient $g_t = -\nabla J(\theta_t)$
- Send $g_t$ to master at iteration $t+\tau$

Algorithm 2 ASGD: master

- Receive gradient $g_t$ from worker $i$ (at iteration $t+\tau$
- Update master’s weights $\theta_{t+\tau+1} = \theta_{t+\tau} - \eta g_t$
- Send parameters $\theta_{t+\tau+1}$ to worker $i$

In other words, gradient $g_t$ is stale if it was computed from parameters $\theta_t$ but applied to $\theta_{t+\tau}$. This gradient staleness is a major obstacle to scaling ASGD: the lag $\tau$ increases as the number of workers $N$ grows, decreasing gradient accuracy, and ultimately reducing the accuracy of the trained model. As a result, ASGD suffers from slow convergence and reduced final accuracy, and may not converge at all if the number of workers is too high [34].

From Lag to Gap. Previous works [34, 4] commonly analyze ASGD staleness using lag $\tau$. We argue that $\tau$ doesn’t reflect the “true” staleness. Therefore, we propose a more precise approach for measuring the staleness, which we call the gap.

Definition 1. We denote $\Delta_{t+\tau} = \theta_{t+\tau} - \theta_t$ and define the gap as:

$$G(\Delta_{t+\tau}) = \text{RMSE}(\Delta_{t+\tau}) = \frac{\|\Delta_{t+\tau}\|_2}{\sqrt{\text{len}(\Delta_{t+\tau})}} = \frac{\|\Delta_{t+\tau}\|_2}{\sqrt{k}}$$

Ideally, there should be no difference between $\theta_{t+\tau}$ and $\theta_t$: when $\Delta_{t+\tau} = 0$, the gradient is computed on the same parameters it will be applied to. This is the case for sequential and synchronous methods such as SGD and SSGD. In asynchronous methods, more workers result in an increased lag $\tau$ and thus a larger gap, as demonstrated in Figure 2(a).

Assumption 1. The gradient of $J$ is an L-Lipschitz continuous function:

$$\|\nabla J(\theta_{t+\tau}) - \nabla J(\theta_t)\|_2 \leq L\|\theta_{t+\tau} - \theta_t\|_2$$

Proposition 1. Under Assumption 1, the accuracy of the stale gradient is bounded by the gap:

$$\|\nabla J(\theta_{t+\tau}) - \nabla J(\theta_t)\|_2 \leq L\|\theta_{t+\tau} - \theta_t\|_2 = L\|\Delta_{t+\tau}\|_2 = L\cdot\sqrt{k}\cdot G(\Delta_{t+\tau})$$

$$\|\nabla J(\theta_{t+\tau}) - \nabla J(\theta_t)\|_2 = O(G(\Delta_{t+\tau}))$$

Figure 1: Gradient staleness in the ASGD training process, adapted from Zheng et al. [35]. Gradient $g_t$ is stale, since it is computed on the approximated future parameters $\hat{\theta}_t$ and applied to $\theta_{t+\tau}$.
Algorithm comparison (all with 8 workers).

Figure 2: The gap between $\theta_{t+\tau}$ and $\theta_t$ while training ResNet-20 on the CIFAR-10 dataset with (a) different numbers of workers, and (b) different asynchronous algorithms. The large drops in $G(\Delta_{t+\tau})$ are caused by the decay of $\eta$ since the gap depends linearly on $\eta$ (Remark 3, Appendix A).

Proposition 1 shows that a larger gap means a larger upper bound on the stale gradient’s accuracy. Conversely, a smaller gap means that the gradient is more accurate. The importance of using the gap instead of the lag is illustrated by a simple example of a worker with $\tau = 2$. If the two updates are exactly in opposite directions and of the same magnitude, the gradient would be accurately computed, as if the lag were zero. However, the lag remains the same ($\tau = 2$), while the gap adjusts according to the two updates and equals to zero, correlating with the gradient’s accuracy.

The Effect of Momentum While the momentum and NAG methods improve SGD convergence rate and accuracy, they make scaling to more workers more difficult. As Figure 2(b) shows, adding momentum to ASGD (NAG-ASGD) increases $G(\Delta_{t+\tau})$, even though the lag $\tau$ is unchanged.

Let $x^i$ be the variable $x$ for worker $i$ (for the master, $i = 0$) and let $x^i_t$ be the value of that variable at iteration $t$. We also denote $\text{prev}(i, t)$ as the last iteration where worker $i$ sent a gradient to the master at time $t$.

**Lemma 1.** For ASGD and NAG-ASGD, $\mathbb{E} [\Delta_{t+\tau}]$ is the sum of gradients\(^1\) and the sum of the momentum vectors respectively:

\[
\mathbb{E} [\Delta_{t+\tau}^{\text{ASGD}}] = -\eta \sum_{i=1}^{N} g_{\text{prev}(i,t+\tau)}^i \\
\mathbb{E} [\Delta_{t+\tau}^{\text{NAG-ASGD}}] = -\eta \sum_{i=1}^{N} v_{\text{prev}(i,t+\tau)}^i
\]

**Assumption 2.** We denote: $a = \sum_{i=1}^{N} g_{\text{prev}(i,t+\tau)}^i$, $b = \gamma \sum_{i=1}^{N} v_{\text{prev}(i,t-1)}^i$. We assume that $\frac{a^2}{\|a\| \cdot \|b\|} \geq \frac{1}{2\|a\|}$. The validity of the assumption is explained in Appendix A.

**Theorem 1.** Under Assumption 2: $G(\mathbb{E} [\Delta_{t+\tau}^{\text{NAG-ASGD}}]) \geq G(\mathbb{E} [\Delta_{t+\tau}^{\text{ASGD}}])$ (proof in Appendix A).

Figure 2(b) demonstrates Theorem 1 empirically: the gap of NAG-ASGD is considerably larger than that of ASGD, even though the lag $\tau$ is unchanged. Intriguingly, maintaining multiple momentum vectors (Multi-ASGD), one for each worker, reduces the gradient staleness compared to a single momentum vector (NAG-ASGD), as shown in Figure 2(b). Recent works [17, 11] show that SGD encourages positive gradient coherence [4] in the training of popular DNNs; thus, consecutive gradient updates tend to have a similar direction. Consecutive gradients update the same momentum vector in NAG-ASGD, whereas in Multi-ASGD they update different momentum vectors. Therefore, the gap is smaller in Multi-ASGD than in NAG-ASGD. Figure 2(b) shows that the gap of both NAG-ASGD and Multi-ASGD is substantially larger than for ASGD without momentum. DANA, detailed in the next section, maintains a small gap throughout training despite using momentum.

4 DANA

DANA is a distributed method that achieves state-of-the-art accuracy even when trained with momentum on large clusters. DANA reduces the gap $G(\Delta_{t+\tau})$ by computing the worker’s gradients

\(^1\)To simplify the analysis, we assume that all workers have equal computation power. This assumption can be relieved by keeping track of the rate of each worker’s updates and weighting them accordingly.
Thus, since the master’s parameters after applying worker $\theta^i$ are computed on and applied to $\Theta^t$, Equation (6) (proof in Appendix A) shows the Bengio-NAG update rule, where the gradient is substituted $\theta^i$ for $\gamma v^{t-1}$.

Bengio et al. [2] proposed a simplified variation of NAG, known as Bengio-NAG. This variation is typically used in deep learning frameworks [21], since it simplifies the implementation. Bengio-NAG defines a new variable $\Theta^t$ to stand for $\theta$ after the momentum update:

$$\Theta^t = \theta^t - \eta \gamma v^{t-1}$$

(5)

Substituting $\theta$ with $\Theta^t$ in the NAG update rule yields the Bengio-NAG update rule:

$$\Theta^{t+1} = \Theta^t - \eta (\gamma v^t + \nabla J(\Theta^t))$$

(6)

Equation (6) (proof in Appendix A) shows the Bengio-NAG update rule, where the gradient is both computed on and applied to $\Theta$, rather than computed on $\tilde{\theta}$ but applied to $\theta$. Therefore, an implementation of NAG requires to store only one set of parameters $\Theta$ in memory.

4.1 The DANA-Zero Update Rule

In DANA-Zero, the master maintains a separate momentum vector $v^i$ for each worker $i$, which is updated with the worker’s gradient $g^i$ using the same update rule as in classic SGD with momentum (Equation (2)). Since the master updates each $g^i$ only with the gradients from worker $i$, we can apply look-ahead using the most recent momentum vectors of the other workers. Based on similar insights from NAG [20], worker $i$’s update moves the master’s parameters $\theta^i$ by $-\eta (\gamma v^i_{prev(i,t-1)} + g^i)$. Thus, since $g^i_{prev}$ is unknown beforehand, $\theta^i - \eta \gamma v^i_{prev(i,t-1)}$ is a good future position approximation of the master’s parameters after applying worker $i$’s momentum vector. By accounting for the current momentum vectors of all workers, DANA estimates the master’s future parameter’s position. Instead of sending the master’s current parameters $\theta^i$, DANA-Zero sends the estimated future position of the master’s parameters after the next $N$ updates, one for each worker:

$$\hat{\Theta}^i_{DANA} \triangleq \theta^i - \eta \gamma \sum_{j=1}^{N} v^j_{i,t-1}$$

(4)

Algorithm 3 DANA-Zero: master

Receive gradient $g^i$ from worker $i$
Update worker’s momentum $v^i \leftarrow \gamma v^i + g^i$
Update master’s weights $\theta^i \leftarrow \theta^i - \eta \gamma v^i$
Send estimate $\hat{\theta} = \theta^i - \eta \gamma \sum_{j=1}^{N} v^j$ to worker $i$

Algorithm 4 DANA-Slim: worker $i$

Receive parameters $\Theta^i$ from master
Compute gradient $g^i \leftarrow \nabla J(\Theta^i)$
Update momentum $v^i \leftarrow \gamma v^i + g^i$
Send update step $\gamma v^i + g^i$ to master

Algorithm 3 shows the DANA-Zero master algorithm; the worker is the same as in ASGD (Algorithm 1). The rationale behind DANA is demonstrated by Lemma 2:

**Lemma 2.** $E[\Delta DANA^{t+\tau}] = E[\Delta ASGD^{t+\tau}]$

**Theorem 2.** Under Assumption 2: $G(E[\Delta DANA^{t+\tau}]) = G(E[\Delta ASGD^{t+\tau}]) \leq G(E[\Delta NAG-ASGD^{t+\tau}])$

Figure 2(b) demonstrates Theorem 2 empirically: despite using momentum, DANA-Zero maintains a small gap throughout the training process, even lower than ASGD. DANA-Zero converges faster than ASGD, resulting in smaller gradients (Appendix D), and therefore a smaller gap.

**Remark 2.** Using a single worker DANA-Zero reduces to a single NAG optimizer (Appendix A).

4.2 Optimizing DANA

In DANA-Zero, the master maintains a momentum vector for every worker, and must also compute $\hat{\theta}$ at each iteration. This adds a computation and memory overhead to the master. DANA-Slim is a variation of DANA-Zero that obtains the same look-ahead as DANA-Zero but without the overhead.

**Bengio-NAG** Bengio et al. [2] proposed a simplified variation of NAG, known as Bengio-NAG. This variation is typically used in deep learning frameworks [21], since it simplifies the implementation. Bengio-NAG defines a new variable $\Theta$ to stand for $\theta$ after the momentum update:

$$\Theta^t \triangleq \theta^t - \eta \gamma v^{t-1}$$

(5)

Substituting $\theta$ with $\Theta^t$ in the NAG update rule yields the Bengio-NAG update rule:

$$\Theta^{t+1} = \Theta^t - \eta (\gamma v^t + \nabla J(\Theta^t))$$

(6)

Equation (6) (proof in Appendix A) shows the Bengio-NAG update rule, where the gradient is both computed on and applied to $\Theta$, rather than computed on $\tilde{\theta}$ but applied to $\theta$. Therefore, an implementation of NAG requires to store only one set of parameters $\Theta$ in memory.

\[\text{See Footnote 1 in Page 4.}\]
**The DANA-Slim Update Rule** We leverage the ideas of Bengio-NAG to optimize DANA.

**Definition 2.** We re-define $\Theta_t$ as $\theta_t$ after applying the momentum update from all future workers. Therefore, $\Theta_{t+1}$ is $\theta_t$ after the current worker’s $i$ update:

$$\Theta_t \triangleq \theta_t - \eta \gamma \sum_{j=1}^{N} v^j_{\text{prev}(j,t-1)}, \quad \Theta_{t+1} = \theta_{t+1} - \eta \gamma \left( v^i_t + \sum_{j \neq i} v^j_{\text{prev}(j,t-1)} \right)$$

The update of each momentum vector remains the following:

$$v^i_t = \gamma v^i_{\text{prev}(i,t-1)} + \nabla J(\Theta_{\text{prev}(i,t)})$$

Which leads to the update rule (see proof as a part of Theorem 3 in Appendix A):

$$\Theta_{t+1} = \Theta_t - \eta \left( \gamma v^i_t + \nabla J(\Theta_{\text{prev}(i,t)}) \right)$$

**Theorem 3.** Substituting $\theta_t$ with $\Theta_t$ in DANA-Zero eliminates the overhead at the master.

Algorithm 4 shows DANA-Slim: the variation of DANA-Zero that uses Bengio-NAG to eliminate the overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim is equivalent to DANA-Zero in all other ways, and provides the same benefits: it uses the same hyperparameters as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. DANA-Slim only changes the worker side and uses the same master algorithm as in ASGD (Algorithm 2); hence, it alleviates the additional overhead at the master. **4.3 Delay Compensation**

Zheng et al. [35] proposed Delay Compensated ASGD (DC-ASGD), a unique approach that tackles the problem of stale gradients, by adjusting the gradient with a second-order Taylor expansion. Due to the high computation and space complexity of the Hessian matrix, they propose a cheap yet effective Hessian approximator that is only based on previous gradients, without the necessity of directly computing the Hessian matrix. We denote $\odot$ as matrix element-wise multiplication.

$$g_t = \nabla J(\theta_t), \quad \hat{g}_t = g_t + \lambda g_t \odot (\theta_{t+\tau} - \theta_t)$$

Equation (9) shows the modification of DC-ASGD (Algorithm 8 in Appendix B.1). The delay compensation, $\lambda g_t \odot (\theta_{t+\tau} - \theta_t)$, adjusts the gradient $g_t$ as if it were computed on $\theta_{t+\tau}$ instead of $\theta_t$; thus, mitigating the gradient staleness. Taylor expansion is more accurate when the source $\theta_t$ is in close vicinity of the approximation point $\theta_{t+\tau}$ (a small gap). Momentum increases $\Delta_{t+\tau} = \theta_{t+\tau} - \theta_t$; therefore, reducing the effectiveness of the delay compensation. DANA-Zero ensures that RMSE($\Delta_{t+\tau}$) is kept small throughout training, even when training with momentum, thereby increasing the effectiveness of the delay compensation. DANA-Zero incorporates the delay compensation, thus further mitigating the gradient staleness. We call this combined method DANA with Delay Compensation (DANA-DC). See Algorithm 9 in Appendix B.1.

5 Experiments

In this section, we present our evaluations and insights regarding DANA. We simulated multiple distributed workers to measure the final test error and convergence speed of different cluster sizes. Since one of our goals in these experiments is to verify that mitigating the gap leads to a better final test error and convergence rate, especially when scaling to more workers, we used the same hyperparameters across all the tested algorithms (see Appendix B.4). These hyperparameters are the original hyperparameters suggested by the authors of each neural network architecture’s respective paper, which are tuned for a single worker. We strengthen our case by comparing to DC-ASGD with parameters which were tuned for 8 workers as suggested by Zheng et al. [35]. We simulate the workers’ execution time using a gamma-distributed model (Appendix B.3), where the execution time for each individual batch is drawn from a gamma distribution. The gamma distribution is a well-accepted model for task execution time, which gives rise to stragglers naturally. The importance of asynchronous training over synchronous training is explained in Appendix C.4.

\(^3\)A single worker may not be a single GPU. DANA, like all ASGD algorithms, can treat each machine with multiple GPUs as a single worker. For example, DANA can run on 32 workers with 8 GPUs each, where each worker performs SSGD internally (which is transparent to the ASGD algorithm).
Algorithms Our evaluations, which are performed with the baseline’s hyperparameters (Appendix B.4) unless stated otherwise, consist of the following algorithms (Appendix B.1):

- **Baseline**: single worker with the same hyperparameters as in the respective NN paper.
- **NAG-ASGD**: asynchronous SGD, which uses a single NAG optimizer for all workers.
- **Multi-ASGD**: asynchronous SGD, which maintains a separate NAG optimizer for each worker.
- **DC-ASGD**: as described in Section 4.3. We set $\gamma = 0.95$ as suggested by Zheng et al. [35].
- **DANA-Slim**: as described in Section 4.2.
- **DANA-DC**: as described in Section 4.3. We set $\lambda = 2$, as suggested by Zheng et al. [35].

5.1 Evaluation on CIFAR

In the CIFAR experiments, bold lines show the mean over the 5 different runs, while transparent bands show the standard deviation. The baseline is the mean of 5 different runs with a single worker.

Figure 3 shows that DANA-DC’s final test error remains similar to the baseline error, without re-tuning the hyperparameters, using up to 24 workers in Figures 3(a) and 3(b) and up to 16 workers in Figure 3(c). Moreover, DANA-DC’s final error is lower than the other algorithms for any number of workers. CIFAR’s final accuracies are listed in Tables 1 to 3 (Appendix C.1).

NAG-ASGD demonstrates how gradient staleness is exacerbated by momentum. NAG-ASGD yields good accuracy with few workers, but fails to converge when trained with more than 16 workers. Multi-ASGD serves as an ablation study: its poor scalability demonstrates that it is not sufficient to simply maintain a momentum vector for every worker. Hence, DANA (Section 4) is also required to achieve fast convergence and low test error.

Figure 4 shows the mean and standard deviation of the test error throughout the training of the different algorithms when trained on 8 workers. This figure demonstrates the significantly better convergence rate of DANA-DC. It is usually similar to the baseline or even faster and it outperforms all the other algorithms. It is noteworthy that DANA-DC’s convergence rate surpasses that
5.2 Evaluation on ImageNet

Figure 5(a) compares final test errors when training the ResNet-50 architecture [9] on ImageNet. DANA-DC remains very close to the baseline and outperforms all the other algorithms in both final test accuracy and convergence speed. Table 4 (Appendix C.2) lists ImageNet’s final test accuracies.

5.3 The Importance of the Gap

Figures 3 and 4 are highly correlated with Figure 2(b), empirically proving that algorithms which maintain a lower gap converge faster and achieve a higher test accuracy. The algorithms in Figure 2(b) share the same lag; therefore, we conclude that the gap is more informative than the lag and that gap mitigation is paramount to asynchronous training. We note that throughout our evaluations, DANA’s gap is an order of magnitude smaller than NAG-ASGD’s, as shown in Figure 2(b).

6 Related Work

Asynchronous training causes gradient staleness, which hinders the convergence. Several approaches [4, 34, 36] proposed to mitigate the gradient staleness by tuning the learning rate with regard to the lag \( \tau \). These approaches, however, are designed for SGD without momentum, and therefore do not address the massive gap that momentum generates. Mitliagkas et al. [19] show that asynchronous training induces implicit momentum, thus the momentum coefficient \( \gamma \) must be decayed when scaling up the cluster size. By mitigating the gap caused by momentum, we prove empirically that, in an asynchronous environment, fast convergence and high final test accuracy is possible, even when \( \gamma \) is relatively high.

Other approaches for mitigating gradient staleness include DC-ASGD [35], which uses a Taylor expansion to mitigate the gradient staleness (Section 4.3). DC-ASGD achieves high accuracy on small clusters, but it falls short when trained on large clusters (Figure 3). Elastic Averaging SGD (EASGD) by Zhang et al. [33] is an asynchronous algorithm that uses a center force to pull the workers’ parameters towards the master’s parameters. This allows each worker to train asynchronously and synchronize with the master once every few iterations. Very recently, Lian et al. [13] proposed AD-PSGD, a decentralized asynchronous approach to scaling SGD that eliminates the parameter server entirely. Not only are these approaches compatible with (and indeed orthogonal to) DANA, but we show that DANA may even amplify the effectiveness of the other approaches, as we demonstrate with DANA-DC (Section 4.3).
7 Conclusion

In this paper we tackle gradient staleness, one of the main difficulties in scaling SGD to more workers in an asynchronous environment. We argue that the lag, commonly used in previous works, does not reflect the “true” staleness. Therefore, we propose a more precise staleness measure, which we call the gap. Based on this new measure, we propose DANA: a novel asynchronous distributed technique that mitigates the gradient staleness by computing the gradient on an estimated future position of the model’s parameters. Thus, DANA efficiently scales to large clusters, despite using momentum, while maintaining high final accuracy and fast convergence. We further introduce DANA-DC to demonstrate that DANA amplifies a delay compensation mechanism, thereby improving the gradients’ accuracy. Throughout our evaluations, DANA-DC consistently outperformed the other methods in both final test error and convergence rate. As for future work, we plan on adapting DANA to recent optimizers, such as Nadam [6], as well as to more recent asynchronous algorithms, in particular AD-PSGD [13] and EASGD [33].

References

[1] S. Ali, H. J. Siegel, M. Maheswaran, S. Ali, and D. Hensgen. Task execution time modeling for heterogeneous computing systems. In Proceedings of the 9th Heterogeneous Computing Workshop, HCW ’00, pages 185–199, 2000. ISBN 0-7695-0556-2.
[2] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. In IEEE International Conference on Acoustics, Speech and Signal Processing, pages 8624–8628, 2013.
[3] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar. signSGD: Compressed optimisation for non-convex problems. In J. Dy and A. Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 560–569, Stockholm Sweden, 2018. PMLR.
[4] W. Dai, Y. Zhou, N. Dong, H. Zhang, and E. Xing. Toward understanding the impact of staleness in distributed machine learning. In International Conference on Learning Representations, 2019.
[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems, pages 1232–1240, 2012.
[6] T. Dozat. Incorporating Nesterov momentum into Adam. ICLR Workshop, 2016.
[7] G. Goh. Why momentum really works. Distill, 2017. doi: 10.23915/distill.00006. URL http://distill.pub/2017/momentum.
[8] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training imagenet in 1 hour. CoRR, abs/1706.02677, 2017.
[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.
[10] G. E. Hinton. Learning multiple layers of representation. Trends in Cognitive Sciences, 11 (10):428–434, 2007.
[11] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In Advances in Neural Information Processing Systems, pages 6389–6399, 2018.
[12] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In OSDI, volume 14, pages 583–598, 2014.
[13] X. Lian, W. Zhang, C. Zhang, and J. Liu. Asynchronous decentralized parallel stochastic gradient descent. In Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 3043–3052. PMLR, 2018.
[14] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In International Conference on Learning Representations, 2018.
[15] N. Loizou and P. Richtárik. Linearly convergent stochastic heavy ball method for minimizing generalization error. *CoRR*, abs/1710.10737, 2017.

[16] N. Loizou and P. Richtárik. Momentum and stochastic momentum for stochastic gradient, newton, proximal point and subspace descent methods. *CoRR*, abs/1712.09677, 2017.

[17] E. Lorch. Visualizing deep network training trajectories with pca. In *ICML Workshop on Visualization for Deep Learning*, 2016.

[18] H. Mikami, H. Suganuma, P. U.-Chupala, Y. Tanaka, and Y. Kageyama. Imagenet/resnet-50 training in 224 seconds. *CoRR*, abs/1811.05233, 2018. URL http://arxiv.org/abs/1811.05233.

[19] I. Mitliagkas, C. Zhang, S. Hadjis, and C. Ré. Asynchrony begets momentum, with an application to deep learning. In *54th Annual Allerton Conference on Communication, Control, and Computing*, pages 693–1004, 2016.

[20] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[21] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques*, 2017.

[22] B. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17, 1964.

[23] 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*, pages 693–701, 2011.

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529 7587:484–9, 2016.

[26] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013.

[27] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems*, pages 1509–1519, 2017.

[28] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015.

[29] M. Yamazaki, A. Kasagi, A. Tabuchi, T. Honda, M. Miwa, N. Fukumo, T. Tabaru, A. Ike, and K. Nakashima. Yet another accelerated SGD: resnet-50 training on imagenet in 74.7 seconds. *CoRR*, abs/1903.12650, 2019. URL http://arxiv.org/abs/1903.12650.

[30] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng. Image classification at supercomputer scale. *CoRR*, abs/1811.06992, 2018. URL http://arxiv.org/abs/1811.06992.

[31] S. Zagoruyko and N. Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, 2016.

[32] H. Zhang, C.-J. Hsieh, and V. Akella. Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent. In *IEEE 16th International Conference on Data Mining*, pages 629–638. IEEE, 2016.

[33] S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, pages 685–693, 2015.
[34] W. Zhang, S. Gupta, X. Lian, and J. Liu. Staleness-aware async-SGD for distributed deep learning. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, pages 2350–2356, 2016.

[35] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z. Ma, and T. Liu. Asynchronous stochastic gradient descent with delay compensation. In Proceedings of the 34th International Conference on Machine Learning, pages 4120–4129, 2017.

[36] 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 J. Dy and A. Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 5970–5979, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.
A Omitted Proofs

Remark 1  The difference between the updated parameters $\theta_{t+1}$ and the approximated future position $\hat{\theta}_t$ is only affected by the newly computed gradient $g_t$, and not by $v_t$.

Proof.

$$\theta_{t+1} - \hat{\theta}_t = \theta_t - \eta v_t - \theta_t + \eta \gamma v_{t-1} = \eta \gamma v_{t-1} - \eta (\gamma v_{t-1} + g_t) = -\eta g_t$$

$\square$

Lemma 1  For ASGD and NAG-ASGD, $\Delta_{t+\tau}$ is the sum of gradients and the sum of the momentum vector respectively:

$$\mathbb{E}[\Delta_{t+\tau}]^{\text{ASGD}} = -\eta \sum_{i=1}^{N} g_{\text{prev}(i,t+\tau)} \quad \mathbb{E}[\Delta_{t+\tau}]^{\text{NAG-ASGD}} = -\eta \sum_{i=1}^{N} v_{\text{prev}(i,t+\tau)}$$

Proof.

$$\mathbb{E}[\Delta_{t+\tau}]^{\text{ASGD}} = \mathbb{E}[\theta_{t+\tau}] - \mathbb{E}[\theta_t] = \left(\theta_t - \eta \sum_{i=1}^{N} g_{\text{prev}(i,t+\tau)}\right) - \theta_t = -\eta \sum_{i=1}^{N} g_{\text{prev}(i,t+\tau)}$$

$$\mathbb{E}[\Delta_{t+\tau}]^{\text{NAG-ASGD}} = \mathbb{E}[\theta_{t+\tau}] - \mathbb{E}[\theta_t] = \left(\theta_t - \eta \sum_{i=1}^{N} v_{\text{prev}(i,t+\tau)}\right) - \theta_t = -\eta \sum_{i=1}^{N} v_{\text{prev}(i,t+\tau)}$$

$\square$

Assumption 2  We denote: $a = \sum_{i=1}^{N} g_{\text{prev}(i,t+\tau)}$, $b = \gamma \sum_{i=1}^{N} v_{\text{prev}(i,t-1)}$. We assume that

$$\frac{a^T b}{\|a\| \cdot \|b\|} \geq -\frac{\|b\|}{2\|a\|}.$$

This assumption means that the angle between the current and past gradients, is lower bounded. The notion of this assumption is validated by empirical results that were shown by Lorch [17] and Li et al. [11].

In order to prove Theorem 1, first we prove the following lemma:

Lemma 3. Under Assumption 2 the following holds:

$$\|a + b\| \geq \|a\|$$

Proof. Starting from Assumption 2 we get:

$$\frac{a^T b}{\|a\| \cdot \|b\|} \geq -\frac{\|b\|}{2\|a\|}$$

$$2a^T b \geq -\|b\|^2$$

$$\|a\|^2 + 2a^T b + \|b\|^2 \geq \|a\|^2$$

$$\|a + b\|^2 \geq \|a\|^2$$

$$\|a + b\| \geq \|a\|$$

$\square$

Theorem 1  Under Assumption 2, momentum expands the gap $G(\Delta_{t+\tau})$ of ASGD: $G(\mathbb{E}[\Delta_{t+\tau}]^{\text{ASGD}}) \geq G(\mathbb{E}[\Delta_{t+\tau}]^{\text{NAG-ASGD}})$.

Proof.

$$G(\mathbb{E}[\Delta_{t+\tau}]^{\text{NAG-ASGD}}) \geq \frac{\eta}{\sqrt{K}} \cdot \left\| \sum_{i=1}^{N} g_{\text{prev}(i,t+\tau)} + \gamma \sum_{i=1}^{N} v_{\text{prev}(i,t-1)} \right\|$$

$$\geq \frac{\eta}{\sqrt{K}} \cdot \left\| \sum_{i=1}^{N} g_{\text{prev}(i,t+\tau)} \right\| = \eta \cdot G\left(\sum_{i=1}^{N} g_{\text{prev}(i,t+\tau)}\right) \geq G(\mathbb{E}[\Delta_{t+\tau}]^{\text{ASGD}})$$

$$\square$$
Remark 3. The $G(\Delta_{t+\tau})$ depends linearly on the learning rate.

**Proof.** Using Lemma 1 we get:

$$G(\Delta_{t+\tau}) = G(-\eta\sum_{i=1}^{N} v_{prev(i,t+\tau)}) = \eta \cdot G(\sum_{i=1}^{N} v_{prev(i,t+\tau)})$$

Lemma 2

$$\mathbb{E} [\Delta_{t+\tau}] = \mathbb{E} [\Delta_{t+\tau}^{\text{ASGD}}]$$

**Proof.**

$$
\mathbb{E} [\Delta_{t+\tau}] = \mathbb{E} [\theta_{t+\tau}] - \mathbb{E} [\hat{\theta}] = \theta_t - \eta \sum_{i=1}^{N} \left( \gamma v^i_{prev(i,t-1)} + g^i_{prev(i,t+\tau)} \right) - \left( \theta_t - \eta \gamma \sum_{i=1}^{N} v^i_{prev(i,t-1)} \right)
$$

$$= -\eta \sum_{i=1}^{N} \left( \gamma v^i_{prev(i,t-1)} + g^i_{prev(i,t+\tau)} - \gamma v^i_{prev(i,t-1)} \right)
$$

$$= -\eta \sum_{i=1}^{N} g^i_{prev(i,t+\tau)} = \mathbb{E} [\Delta_{t+\tau}^{\text{ASGD}}]
$$

Theorem 2

$$G(\mathbb{E} [\Delta_{t+\tau}^{\text{DANA}}]) = G(\mathbb{E} [\Delta_{t+\tau}^{\text{ASGD}}]) \leq G(\mathbb{E} [\Delta_{t+\tau}^{\text{NAG-ASGD}}])$$

**Proof.** Applying the gap on Lemma 2 we get:

$$G(\mathbb{E} [\Delta_{t+\tau}^{\text{DANA}}]) = G(\mathbb{E} [\Delta_{t+\tau}^{\text{ASGD}}])$$

Then using Theorem 1 we get:

$$G(\mathbb{E} [\Delta_{t+\tau}^{\text{DANA}}]) = G(\mathbb{E} [\Delta_{t+\tau}^{\text{ASGD}}]) \leq G(\mathbb{E} [\Delta_{t+\tau}^{\text{NAG-ASGD}}])$$

Remark 2

DANA-Zero Equivalence to Nesterov. When running with one worker ($N = 1$) DANA-Zero reduces to a single NAG optimizer. This can be shown by merging the worker and master (Algorithms 1 and 3) into a single algorithm: since at all times $\theta_t = \theta_0 - \eta \gamma v_{t-1}$, the resulting algorithm trains one set of parameters $\theta$, which is exactly the Nesterov update rule. Algorithm 5 shows the combined algorithm, equivalent to the standard NAG optimizer.

**Algorithm 5**

Fused DANA-Zero (when $N = 1$)

- Compute gradient $g_t \leftarrow \nabla J(\theta_t - \eta \gamma v_{t-1})$
- Update momentum $v_t \leftarrow \gamma v_{t-1} + g_t$
- Update weights $\theta_{t+1} \leftarrow \theta_t - \eta v_t$
**Equation (6)** The equivalence of Bengio-NAG to vanilla NAG.

\[
\begin{align*}
\theta_{t+1} &= \theta_t - \eta v_t \\
\downarrow & \quad \text{(Equation (5))}
\end{align*}
\]

\[
\begin{align*}
\Theta_{t+1} + \eta \gamma v_t &= \Theta_t + \eta \gamma v_{t-1} - \eta v_t \\
\downarrow & \quad \Theta_{t+1} + \eta \gamma v_t = \Theta_t + \eta \gamma v_{t-1} - \eta(\gamma v_{t-1} + \nabla J(\Theta_t)) = \Theta_t - \eta \nabla J(\Theta_t) \\
\downarrow & \quad \Theta_{t+1} = \Theta_t - \eta (\gamma v_t + \nabla J(\Theta_t))
\end{align*}
\]

**Theorem 3** Substituting \(\theta_t\) with \(\Theta_t\) in DANA-Zero, eliminates the overhead at the master.

**Proof.** We start by proving the DANA-Slim update rule (Equation (8)):

\[
\begin{align*}
\theta_{t+1} &= \theta_t - \eta v^i_t \\
\downarrow & \quad \text{(Definition 2)}
\end{align*}
\]

\[
\begin{align*}
\Theta_{t+1} + \eta \gamma \left(v^i_t + \sum_{j \neq i}^N v^j_{prev(j,t-1)}\right) &= \Theta_t + \eta \gamma \sum_{j=1}^N v^j_{prev(j,t-1)} - \eta v^i_t \\
\downarrow & \quad \Theta_{t+1} = \Theta_t + \eta \gamma \left(v^i_{prev(i,t-1)} - \left(1 + \frac{1}{\gamma}\right) \cdot v^i_t\right) \\
\downarrow & \quad \Theta_{t+1} = \Theta_t - \eta (\gamma v^i_t + \nabla J(\Theta_{prev(i,t)}))
\end{align*}
\]

Which means that using \(\Theta_t\), the update of the master only uses the current master’s parameters and worker \(i\)’s current momentum vector. Since the master next sends worker \(i\) the new parameters \(\Theta_{t+1}\), the master doesn’t need to “remember” all the momentum vectors of the other workers and the master’s algorithm remains unchanged from the ASGD algorithm (Algorithm 2). This proves there is no added overhead at the master.

**B Experimental Setup**

**B.1 Algorithms**

Algorithms 6 to 9 only change the master’s algorithm; the complementary worker algorithm is the same as ASGD (Algorithm 1). The master’s scheme is a simple FIFO. We consider parameter server optimizations to be beyond the scope of this paper.

**Algorithm 6 NAG-ASGD: master**

Receive gradient \(g^i\) from worker \(i\)
Update momentum \(v \leftarrow \gamma v + g^i\)
Update master’s weights \(\theta^0 \leftarrow \theta^0 - \eta v\)
Send \(\theta^0\) to worker \(i\)

**Algorithm 7 Multi-ASGD: master**

Receive gradient \(g^i\) from worker \(i\)
Update momentum \(v^i \leftarrow \gamma v^i + g^i\)
Update master’s weights \(\theta^0 \leftarrow \theta^0 - \eta v^i\)
Send \(\theta^0\) to worker \(i\)
Algorithm 8 DC-ASGD: master

Receive gradient \( g^i \) from worker \( i \)
Update the gradient according to the delay compensation term \( \hat{g}^i = g^i + \lambda g^i \odot g^i \odot (\theta^0 - \theta^i) \)
Update momentum \( v^i \leftarrow \gamma v^i + \hat{g}^i \)
Update master’s weights \( \theta^0 \leftarrow \theta^0 - \eta v^i \)
Send \( \theta^0 \) to worker \( i \)

Algorithm 9 DANA-DC: master

Receive gradient \( g^i \) from worker \( i \)
Update the gradient according to the delay compensation term \( \hat{g}^i = g^i + \lambda g^i \odot g^i \odot (\theta^0 - \theta^i) \)
Update momentum \( v^i \leftarrow \gamma v^i + \hat{g}^i \)
Update master’s weights \( \theta^0 \leftarrow \theta^0 - \eta v^i \)
Send estimate \( \hat{\theta} = \theta^0 - \eta \gamma \sum_{j=1}^{N} v^j \) to worker \( i \)

B.2 Datasets

CIFAR The CIFAR-10 [10] dataset is comprised of 60k RGB images partitioned into 50k training images and 10k test images. Each image contains 32x32 RGB pixels and belongs to one of ten equal-sized classes. CIFAR-100 is similar but has 100 classes. Link.

ImageNet The ImageNet dataset [24], known as ILSVRC2012, consists of RGB images, each labeled as one of 1000 classes. Images are partitioned to 1.28 million training images and 50k validation images, and each image is randomly cropped and re-sized to 224x224 (1-crop validation). Link.

B.3 Gamma Distribution

Ali et al. [1] suggest a method called CVB to simulate the run-time of a distributive network of computers. The method is based on several definitions:

Definition 3. Task execution time variables:
- \( \mu_{\text{task}} \) - mean time of tasks
- \( \nu_{\text{task}} \) - variance of tasks
- \( \mu_{\text{mach}} \) - mean computation power of machines
- \( \nu_{\text{mach}} \) - variance of computation power of machines
- \( \alpha_{\text{task}} = \frac{1}{\nu_{\text{task}}^2} \)
- \( \alpha_{\text{mach}} = \frac{1}{\nu_{\text{mach}}^2} \)

\( G(\alpha, \beta) \) is a random number generated using a gamma distribution where \( \alpha \) is the shape and \( \beta \) is the scale.

Since in our case all tasks are similar and run on a batch size of \( B \), the algorithm for deciding the execution-time of every task on a certain machine is reduced to one of the following:

Algorithm 10 Task execution time - homogeneous machines

\[
\beta_{\text{task}} = \frac{\mu_{\text{task}}}{\alpha_{\text{task}}} \\
q = G(\alpha_{\text{task}}, \beta_{\text{task}}) \\
\beta_{\text{mach}} = \frac{q}{\alpha_{\text{mach}}} \\
\text{for } i \text{ from } 0 \text{ to } T - 1: \\
\text{time} = G(\alpha_{\text{mach}}, \beta_{\text{mach}})
\]
Algorithm 11 Task execution time - heterogeneous machines

\[ \beta_{mach} = \frac{\mu_{mach}}{\sigma_{mach}} \]

for j from 0 to M:

\[ p[j] = G(\alpha_{mach}, \beta_{mach}) \]

\[ \beta_{task}[j] = \frac{p[j]}{\alpha_{task}} \]

for i from 0 to \( T - 1 \):

\[ time = G(\alpha_{task}, \beta_{task[curr]}) \]

where \( T \) is the total amount of tasks of all the machines combined (the total number of batch iterations), \( M \) is the total number of machines (workers) and \( curr \) is the machine currently about to run.

We note that algorithms Algorithms 10 and 11, both naturally give rise to stragglers. In the homogeneous algorithm, all workers have the same mean execution time but some tasks can still be very slow (which generally means that in every epoch a different machine will be the slowest). In the heterogeneous algorithm, every machine has a different mean execution time throughout the training. We further note that \( p[j] \) is the mean execution time of machine \( j \) on the average task.

In our experiments we simulated execution times using the following parameters as suggested by Ali et al. [1]: \( \mu_{task} = \mu_{mach} = B \cdot V_{mach}^2 \), where \( B \) is the batch size, yielding a mean execution time of \( \mu \) simulated time units which is proportionate to \( B \). In the homogeneous setting \( V_{mach} = 0.1 \), whereas in the heterogeneous setting \( V_{task} = 0.1 \).

B.4 Hyperparameters

Since one of our intentions in these experiments is to verify that mitigating the gap leads to a better final test error and convergence rate, especially when scaling to more workers, we used the same hyperparameters across all the tested algorithms. These hyperparameters are the original hyperparameters of the respective neural network architecture, which are tuned for a single worker.

CIFAR-10 ResNet-20
- Initial Learning Rate \( \eta \): 0.1
- Momentum Coefficient \( \gamma \): 0.9 with NAG
- Dampening: 0 (no dampening)
- Batch Size \( B \): 128
- Weight Decay: \( 1e-4 \)
- Learning Rate Decay: 0.1
- Learning Rate Decay Schedule: Epochs 80 and 120
- Total Epochs: 160

CIFAR-10/100 Wide ResNet 16-4
- Initial Learning Rate \( \eta \): 0.1
- Momentum Coefficient \( \gamma \): 0.9 with NAG
- Dampening: 0 (no dampening)
- Batch Size \( B \): 128
- Weight Decay: \( 5e-4 \)
- Learning Rate Decay: 0.2
- Learning Rate Decay Schedule: Epochs 60, 120 and 160
- Total Epochs: 200

ImageNet ResNet-50
- Initial Learning Rate \( \eta \): 0.1
- Momentum Coefficient \( \gamma \): 0.9 with NAG
- Dampening: 0 (no dampening)
- Batch Size \( B \): 256
- Weight Decay: \( 1e-4 \)

16
Learning Rate Warm-Up  In the early stages of training, the network changes rapidly, causing error spikes. For all algorithms, we follow the gradual warm-up approach proposed by Goyal et al. [8] to overcome this problem: we divide the initial learning rate by the number of workers $N$ and ramp it up linearly until it reaches its original value after 5 epochs. We also use momentum correction [8] in all algorithms to stabilize training when the learning rate changes.

C  Additional Results

C.1  CIFAR Final Accuracies

When reaching 32 workers, DANA-DC starts to show signs of divergence. However, we note that when tuning the learning rate $\eta$, DANA-DC reaches a significantly lower test error than that shown in Tables 1 to 3: when trained on 32 workers with $\eta = 0.025$, DANA-DC reaches a test error of only 2.5% higher than the baseline on both CIFAR10 scenarios and 4.5% higher than the baseline on CIFAR100.

| #Workers | DANA-DC  | DANA-Slim  | DC-ASGD  | Multi-ASGD | NAG-ASGD  |
|----------|----------|------------|----------|------------|-----------|
| 4        | 91.53 ± 0.27 | 91.66 ± 0.08 | 91.66 ± 0.18 | 91.48 ± 0.15 | 91.31 ± 0.16 |
| 8        | 91.39 ± 0.09 | 91.32 ± 0.07 | 91.04 ± 0.35 | 91.33 ± 0.18 | 90.43 ± 0.26 |
| 12       | 91.32 ± 0.35 | 91.38 ± 0.23 | 64.45 ± 13.95 | 90.73 ± 0.09 | 82.87 ± 4.03 |
| 16       | 91.17 ± 0.25 | 91.04 ± 0.19 | 15.47 ± 10.95 | 85.24 ± 1.82 | 18.83 ± 12.06 |
| 20       | 90.81 ± 0.25 | 90.96 ± 0.22 | 10.0 ± 0.0 | 63.95 ± 8.36 | 10.0 ± 0.0 |
| 24       | 89.41 ± 0.3 | 89.92 ± 0.68 | 10.0 ± 0.0 | 24.98 ± 19.45 | 27.24 ± 19.66 |
| 32       | 81.38 ± 4.89 | 79.25 ± 6.2 | 10.0 ± 0.0 | 12.32 ± 4.64 | 25.31 ± 18.9 |

Table 1: ResNet-20 CIFAR10 Final Test Accuracy (Baseline 91.63%)
Table 3: Wide ResNet 16-4 CIFAR100 Final Test Accuracy (Baseline 76.72%)

| #Workers | DANA-DC         | DANA-Slim        | DC-ASGD         | Multi-ASGD      | NAG-ASGD        |
|----------|-----------------|------------------|-----------------|-----------------|-----------------|
|          | ±                | ±                | ±                | ±                | ±               |
| 4        | 76.56 ± 0.25     | 76.34 ± 0.31     | 76.35 ± 0.28    | 76.38 ± 0.32    | 75.84 ± 0.27    |
| 8        | 75.86 ± 0.25     | 76.08 ± 0.17     | 75.38 ± 0.2     | 75.28 ± 0.19    | 74.23 ± 0.43    |
| 12       | 75.5 ± 0.31      | 75.42 ± 0.25     | 73.67 ± 0.16    | 73.65 ± 0.41    | 69.13 ± 0.82    |
| 16       | 74.86 ± 0.1      | 74.88 ± 0.27     | 70.82 ± 0.08    | 71.01 ± 0.51    | 65.39 ± 2.55    |
| 20       | 73.35 ± 0.17     | 73.26 ± 0.31     | 68.8 ± 0.21     | 68.42 ± 0.33    | 36.63 ± 4.75    |
| 24       | 71.63 ± 0.44     | 71.63 ± 0.67     | 65.05 ± 1.32    | 65.2 ± 0.97     | 11.39 ± 6.06    |
| 28       | 69.42 ± 0.9      | 69.18 ± 0.75     | 56.07 ± 3.45    | 54.68 ± 2.47    | 9.65 ± 8.02     |
| 32       | 67.04 ± 0.8      | 67.34 ± 0.86     | 32.09 ± 6.65    | 32.16 ± 6.47    | 5.47 ± 5.56     |

C.2 ImageNet Final Accuracies

Table 4: ResNet-50 ImageNet Final Test Accuracy (Baseline 75.64%)

| #Workers | DANA-DC         | DANA-Slim        | DC-ASGD         | Multi-ASGD      | NAG-ASGD        |
|----------|-----------------|------------------|-----------------|-----------------|-----------------|
|          | ±                | ±                | ±                | ±                | ±               |
| 16       | 75.54%          | 74.95%           | 72.64%          | 74.96%          | 73.22%          |
| 32       | 74.86%          | 74.89%           | 59.99%          | 71.72%          | 70.64%          |
| 48       | 73.80%          | 73.75%           | 31.71%          | 65.13%          | 66.78%          |
| 64       | 73.58%          | 69.88%           | 8.1%            | 54.04%          | 59.81%          |
| 128      | 69.50%          | NaN              | NaN             | NaN             | NaN             |

Table 4 lists the final test accuracy of the different algorithms when training the ResNet-50 architecture [9] on ImageNet. DANA consistently outperforms all the other algorithms.

C.3 CIFAR-10 Distributed Experiments

While this work focuses on improving ASGD accuracy without adding overhead, we also measured speedup, defined as the runtime for DANA-Slim with \( N \) workers divided by the runtime for the single worker baseline.

Figure 6 shows the speedup and final test error when running DANA-Slim on the Google Cloud Platform with a single parameter server (master) and one Nvidia Tesla V100 GPU per machine, when training ResNet-20 on the CIFAR-10 dataset. It shows speedup of up to \( 16 \times \) when training with \( N = 24 \) workers, and as before, its final test error remains close to the baseline for up to \( N = 24 \) workers.

At 24 workers, the parameter server becomes a bottleneck. This phenomenon is consistent with literature [28] on ASGD, and is well-studied. Since the DANA master is unchanged from the ASGD algorithm (Algorithm 2), existing techniques, such as sharding the parameter server [5], improving network utilization [12], lock-free synchronizations [23, 32], and gradient compression [14, 27, 3], are fully compatible with DANA but are beyond the scope of this work.

C.4 Asynchronous Speedup

Cloud computing is becoming increasingly popular as a platform to perform distributed training of deep neural networks. Although synchronous SGD is currently the primary method [18, 30, 29] to distribute the learning process, it suffers from substantial slowdowns when running in non-dedicated environments such as the cloud. This shortcoming is magnified in heterogeneous environments. ASGD addresses SSGD’s drawback and enjoys linear speedup in terms of the number of workers.
Figure 6: DANA speedup (solid line) and final test error (dashed) when training ResNet-20 on CIFAR-10 with different numbers of workers.

Figure 7: Theoretical speedups for DANA (or any ASGD) and SSGD when batch execution times are drawn from a gamma distribution. Communication overheads are not modeled; however, asynchronous algorithms are more communication efficient. Therefore, modeling the communication overheads should expand the gap between the asynchronous and synchronous training.

(a) Async (DANA, ASGD) and sync (SSGD) speedup. (b) DANA speedup over SSGD.

in both heterogeneous and homogeneous environments. This makes ASGD a potentially better alternative for cloud computing.

Figure 7(a) shows the theoretically achievable speedup, based on the detailed gamma-distributed model, for asynchronous (DANA and other ASGD variants) and synchronous algorithms. The asynchronous algorithms can achieve linear speedup; the synchronous algorithm (SSGD) falls short as we increase the number of workers, since it must wait after each iteration until all workers complete their batch. Figure 7(b) shows that DANA (or any ASGD variant) is up to 21% faster than SSGD. This speedup is an underestimate, since our simulation only includes batch execution times, and does not model execution time of barriers, all-gather operations, and other overheads.

C.5 Heterogeneous Environment

In the heterogeneous environment experiments, Figure 8(a), the algorithms scale better than in the homogeneous environment experiments (Figure 3(a)). The reason is that stragglers naturally have
less impact on the training process. We will demonstrate this with a toy example. Consider an asynchronous environment with only $N = 2$ workers, where one worker is significantly faster than the other. Therefore, the fast worker will run as in sequential SGD, since its gap and lag will mostly be zero. Conversely, the slow worker will have minimal impact.

This suggests that high accuracy is more easily attainable in asynchronous, heterogeneous environments than in homogeneous environments. Figures 8(a) and 8(b) show that even in a heterogeneous environment DANA-DC converges the fastest and achieves the highest final accuracy. Final accuracies are listed in Table 5 below.

Table 5: Heterogeneous Environment ResNet 20 CIFAR10 Final Test Accuracy (Baseline 91.63%)

| #Workers | DANA-DC   | DANA-Slim | DC-ASGD   | Multi-ASGD | NAG-ASGD |
|----------|-----------|-----------|-----------|------------|----------|
| 4        | 91.57 ± 0.14 | 91.7 ± 0.18 | 91.6 ± 0.14 | 91.77 ± 0.22 | 91.38 ± 0.12 |
| 8        | 91.57 ± 0.18 | 91.55 ± 0.28 | **91.72 ± 0.21** | 91.59 ± 0.11 | 91.15 ± 0.23 |
| 16       | 91.28 ± 0.21 | **91.31 ± 0.17** | 90.98 ± 0.5 | 91.12 ± 0.3 | 83.65 ± 5.17 |
| 24       | **91.21 ± 0.19** | 90.94 ± 0.27 | 90.11 ± 0.92 | 89.6 ± 2.03 | 39.36 ± 36.01 |
| 32       | 90.33 ± 0.58 | **90.52 ± 1.04** | 57.62 ± 38.93 | 74.18 ± 32.1 | 37.52 ± 34.12 |

**D Normalized Gap**

Figure 9(a) shows the gradient norm throughout the training process of different asynchronous algorithms. The gradients of ASGD without momentum are noticeably larger than the algorithms that do use momentum. Lemma 2 proves that ASGD without momentum and DANA-Zero should have a similar gap, when lag $\tau$ is the same. This statement holds true only if the gradients were computed on the same parameters $\theta_t$. However, momentum accelerates the convergence, which leads to
smaller gradients, as shown in Figure 9(a). Therefore, DANA-Zero’s gap is smaller than ASGD’s, as shown in Figure 2(b).

To compare the algorithms when the norm of the gradients is the same, we define the normalized gap as $G^*(\Delta_{t+\tau}) = \frac{G}{\|g\|}$. Figure 9(b) shows that ASGD’s normalized gap is roughly similar to DANA-Zero’s, yet smaller throughout the training process.