SAPipe: Staleness-Aware Pipeline for Data-Parallel DNN Training

Yangrui Chen
The University of Hong Kong
yrchen@cs.hku.hk

Cong Xie
ByteDance
cong.xie@bytedance.com

Meng Ma
ByteDance
meng.ma@bytedance.com

Juncheng Gu
ByteDance
juncheng.gu@bytedance.com

Yanghua Peng
ByteDance
pengyanghua.yanghua@bytedance.com

Haibin Lin
ByteDance
haibin.lin@bytedance.com

Chuan Wu
The University of Hong Kong
cwu@cs.hku.hk

Yibo Zhu
ByteDance
zhuyibo@bytedance.com

Abstract

Data parallelism across multiple machines is widely adopted for accelerating distributed deep learning, but it is hard to achieve linear speedup due to the heavy communication. In this paper, we propose SAPipe, a performant system that pushes the training speed of data parallelism to its fullest extent. By introducing partial staleness, the communication overlaps the computation with minimal staleness in SAPipe. To mitigate additional problems incurred by staleness, SAPipe adopts staleness compensation techniques including weight prediction and delay compensation with provably lower error bounds. Additionally, SAPipe presents an algorithm-system co-design with runtime optimization to minimize system overhead for the staleness training pipeline and staleness compensation. We have implemented SAPipe in the BytePS framework, compatible to both TensorFlow and PyTorch. Our experiments show that SAPipe achieves up to 157% speedups over BytePS (non-stale), and outperforms PipeSGD in accuracy by up to 13.7%.

1 Introduction

Deep Neural Networks (DNNs) have achieved ground-breaking performance on a wide range of domains, such as computer vision (CV) [10, 17] and natural language processing (NLP) [29, 7]. Meanwhile, the model sizes and data volumes have grown exponentially, making DNN training time-consuming and resource-intensive. The most common approach to accelerate DNN training is to use data parallelism, scaling DNN training across multiple devices. Despite the substantial speedup, distributed machine learning systems with data parallelism often cannot fully utilize the computation resources and achieve linear scaling (*i.e.*, GPU number times single-GPU training speed), due to non-negligible communication overhead [31, 2, 23, 13].

Many recent studies have been devoted to developing communication acceleration techniques. Some works reduce communication traffic using gradient compression [2] or mixed-precision training [21],

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

while others schedule communication to overlap it with computation. For example, ByteScheduler [23] and PACE [3] propose preemptive communication scheduling to hide the communication overhead within forward computation time. These communication scheduling approaches reduce the communication overhead without affecting the convergence of training, but still cannot fully hide communication when the communication-to-computation ratio is high.

A new direction to accelerate distributed DNN training has been explored, which intentionally introduces staleness to the training pipeline in order to further increase the overlap between communication and computation. For example, PipeSGD [19] uses the gradients from the previous iteration in stochastic gradient descent (SGD), resulting in fixed 1-step staleness.

Though introducing fixed staleness may fully overlap communication with computation for some DNNs, the staleness may also incur severe problems to the convergence of training. We observe significant accuracy degradation or even training divergence with only 1-step staleness of gradients in our experiments. We identify this key limitation and propose 2 solutions: 1) partial staleness, which introduces staleness to a limited number of layers; 2) staleness compensation, which compensates the 1-step staleness by predicting the gradient to be produced in the next iteration, with optimized implementation to reduce the overhead of prediction.

We design a performant and Staleness-Aware communication Pipeline (SAPipe) system for accelerating distributed DNN training, which reduces the communication overhead in distributed training by overlapping communication with computation, and approaches the linear scaling.

The main contributions of this paper are as follows:

- We propose a partial staleness algorithm, which finds the minimal number of layers to introduce staleness to, so as to keep the training pipeline running without stall (§ 3.1).
- We adopt multiple staleness compensation techniques, including delay compensation, weight prediction and their combinations (§ 3.2).
- We propose an algorithm-system co-design, kernel fusion, and other runtime optimizations in SAPipe, that are especially designed and implemented to minimize the system overhead of partial staleness and staleness compensation (§ 3.3).
- We provide theoretical guarantees to show that SAPipe achieves the same convergence rate as vanilla SGD, and conditionally better error bounds compared to PipeSGD (§ 4).
- We demonstrate that SAPipe outperforms existing frameworks. SAPipe achieves up to 157% speedups over BytePS (non-stale), and outperforms PipeSGD in accuracy by up to 13.7% (§ 5).

2 Background

Preliminaries. In distributed deep learning, we solve the following optimization problem with n workers: $\min_{x\in\mathbb{R}^d} F(x)$, where $F(x) = \frac{1}{n}\sum_{i\in[n]} F_i(x) = \frac{1}{n}\sum_{i\in[n]} \mathbb{E}_{z_i\sim\mathcal{D}_i} f(x; z_i)$, $\forall i \in [n]$, is the objective function, $x \in \mathbb{R}^d$ is the set of model parameters (d is the total number of model parameters), z_i is a mini-batch of data randomly sampled from the local data $\mathcal{D}_i$ on device i, and $f(\cdot)$ is loss function. A typical DNN is composed of m layers, which are concatenated into a flattened vector x for simplicity.

Distributed training with data parallelism. Data parallelism partitions the training data onto multiple devices, *i.e.*, workers. Each worker propagates its local data through the model and calculate the loss (*forward propagation*). It uses the loss value to compute the gradients of each parameter (*backward propagation*), and aggregates them from all workers, before updated to the global model. To facilitate distributed training, the parameter server [18] and all-reduce [26] are the two most popular communication architectures for gradient aggregation. The detailed process is shown in Algorithm 1, highlighted in blue.

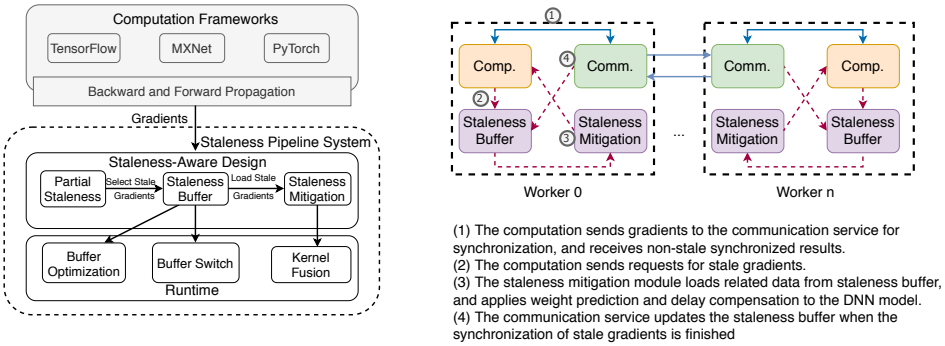
Staleness pipeline. To hide communication time, previous works (*e.g.*, PipeSGD [19]) introduce 1-step staleness to the training pipeline, where the forward computation can progress without waiting for the gradient synchronization. To be more specific, the gradient aggregation of each layer is executed once its backward computation is finished, followed by the optimizer update using the aggregated gradient from the previous iteration. Thus, the gradient aggregation overlaps with not only the backward computation and optimizer update of the current iteration, but also the forward computation of the next iteration. The gradient aggregation initiated in the current iteration will be

Algorithm 1 Distributed Training / Staleness Training Pipeline (PipeSGD)

Initialize x_0
for all iteration $t \in [T]$ **do**
  for all workers $i \in [n]$ in parallel **do**
    Compute $\nabla f(x_{t-1}; z_{i,t})$, $z_{i,t} \sim \mathcal{D}_i$
    if $t = 1$ **then**
      Same as $t > 1$ Pass
    else
      $g_{i,t} \leftarrow \nabla f(x_{t-1}; z_{i,t})$
      $g_{i,t} \leftarrow \nabla f(x_{t-2}; z_{i,t-1})$
      $g_t = \frac{1}{n} \sum_{i \in [n]} g_{i,t}$
      $x_t \leftarrow optimizer\left(x_{t-1}, g_t, \eta_t\right)$

Algorithm 2 Staleness-Aware Pipeline with Delay Compensation (SAPipe-DC)

Initialize x_0
for all iteration $t \in [T]$ **do**
  for all workers $i \in [n]$ in parallel **do**
    Compute $\nabla f(x_{t-1}; z_{i,t})$, $z_{i,t} \sim \mathcal{D}_i$
    if $t > 1$ **then**
      $g_{i,t} = \nabla f(x_{t-2}; z_{i,t-1})$
      $g_t = \frac{1}{n} \sum_{i \in [n]} g_{i,t}$
      $\Delta x_t = x_{t-1} - x_{t-2}$
      $g_t^{DC} \leftarrow DC_\lambda(g_t, \Delta x_t)$
      ▷ DC is a func. defined in Equ. (1)
      $x_t \leftarrow optimizer\left(x_{t-1}, g_t^{DC}, \eta_t\right)$



(a) Architecture (b) Workflow

Figure 1: The architecture and workflow of SAPipe. Solid lines: data flow with all gradients. Dashed lines: data flow with stale gradients.

finished in the next iteration. We outline the conceptual distributed staleness training pipeline in Algorithm 1, highlighted in orange. Staleness pipeline results in delayed gradients, which may cause significant accuracy degradation of the converged model, as shown in our experiments (§ 5).

3 SAPipe Design

To address the convergence issue of staleness pipeline, we design a staleness-aware system, SAPipe, as shown in Figure 1(a), based on the following key designs.

Partial Staleness. Not all layers in a DNN model require a staleness training pipeline to fully hide communication within computation. Our partial staleness algorithm finds a minimal number of layers to be updated by stale gradients (§ 3.1), while updating the other layers without staleness.

Staleness Compensation. To further mitigate the problem caused by staleness of the layers chosen by the partial staleness algorithm, we compensate the staleness of the gradients by multiple approaches including delay compensation and weight prediction (§ 3.2).

Optimized Runtime. Both staleness training pipeline and staleness compensation incur additional overhead of computation and memory copy. To minimize such overhead, we adopt several system optimizations to achieve the high-performance runtime (§ 3.3).

Table 1: Notations

Notation	Description	Notation	Description
b_i	Duration of backward operator i	T, t	Total number and index of iterations
u_i	Duration of forward operator i	g_t	Stochastic gradient $g_t = \frac{1}{n}\sum_{i\in[n]} g_{i,t}$
v_i	Duration of comm. operator i	$(g_t)_j$	The jth coordinate of g_t, $j \in [d]$
c	Completion time of operators	$(g_{i,t})_j$	The jth coordinate of $g_{i,t}$, on worker i
n	Total number of workers	$(\nabla F_t)_j$	The jth coordinate of $\nabla F(x_t)$, $j \in [d]$
m	Total number of layers	$\circ$	Hadamard (coordinate-wise) product
x	Model parameter $x \in \mathbb{R}^d$	d	The number of model parameters

3.1 Partial Staleness

Due to the layer-wise structure of DNN models [1] and the reverse order of forward and backward passes, it only requires parts of the gradients to be stale to hide the communication overhead. Reducing the number of stale gradients mitigates the problems caused by staleness, improving convergence and keeping communication overhead hidden at the same time. We seek to find the minimal number of stale gradients such that there is no training pipeline stall, i.e., no waiting time for computing devices.

A naive way is to enumerate all possible combinations of gradients and check if they lead to training pipeline stall, which involves prohibitive complexity. Fortunately, the execution orders of the layers in sequential models are fixed, which provides the opportunity for efficient searching.

Theorem 1. *In a training pipeline without stall, if some forward layer is stale, then all its preceding forward layers are stale.*

Detailed proof of Theorem 1 is in the Appendix. Then we consider the case that the first k layers need to be trained on staled gradients for fully overlapping its computation and communication, and formulate the following convex program for finding the minimal number of stale layers. Here the variables are defined in Table 1.

$$
\begin{aligned}
\text{minimize} \quad & k \\
\text{subject to} \quad & \sum_i v_i \le \sum_i b_i + \sum_i u_i, \ \sum_{i=k+1}^{m} v_i \le \sum_{i=1}^{m-1} b_i + \sum_{i=1}^{k} u_i, i = 1, \dots, m; \\
& 0 \le k \le m, v_i > 0, b_i > 0, u_i > 0, \ i = 1, \dots, m.
\end{aligned}
$$

The first constraint aims to keep communication and computation fully overlapped. The second constraint ensures that the non-delayed gradients should be synchronized before the start of computation of the corresponding layers. We assume that the computation time of each layer remains the same under different solutions and the execution order follows FIFO. The above problem can be solved in $O(m^2)$ time by enumerating k. The optimal solution gives the minimal delayed gradients and ensures maximal training throughput.

In SAPipe, the first k layers are updated with 1-step staleness, while the remaining layers are updated by vanilla distributed SGD without staleness (see Figure 2). The basic version of SAPipe (referred to as vanila SAPipe) is a mixture of distributed SGD and PipeSGD as given in Algorithm 1. For theoretical analysis, we add the following definition.

Definition 1. *Let $\kappa = \frac{d_{stale}}{d} \in [0, 1]$ denote the portion of model parameters updated by stale gradients, where d_{stale} is the total number of model parameters contained in the k layers that are updated with 1-step staleness.*

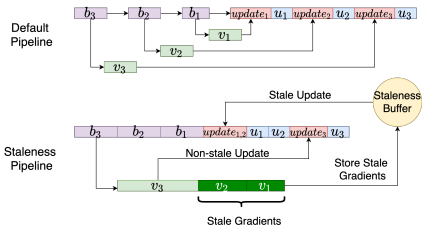


Figure 2: DNN training pipeline. The arrows denote dependencies between two operators.

[1] We focus on sequential models, and the discussion about extending our methods to DAG models can be found at Appendix D.

Algorithm 3 Staleness-Aware Pipeline with Weight Prediction (SAPipe-WP)

1: Initialize x_0
2: **for all** iteration $t \in [T]$ **do**
3: **for all** workers $i \in [n]$ in parallel **do**
4: **if** $t = 1$ **then**
5: Compute $\nabla f(x_0; z_{i,0})$, $z_{i,0} \sim \mathcal{D}_i$
6: **else**
7: $g_{i,t} \leftarrow \nabla f(\tilde{x}_{i,t-1}; z_{i,t-1})$ ▷ Computed in the prev. iteration
8: Option 1: $\tilde{g}_{i,t} \leftarrow g_{i,t}$ ▷ Local gradient
9: Option 2: $\tilde{g}_{i,t} \leftarrow g_{t-1}$ ▷ Latest sync. gradient
10: Option 3: $\tilde{g}_{i,t} \leftarrow DC(g_{t-1} - \frac{1}{n} g_{i,t-1}, \Delta x_t) + \frac{1}{n} g_{i,t}$, where $\Delta x_t = x_{t-1} - x_{t-2}$
11: $\tilde{x}_{i,t} \leftarrow optimizer\,(x_{t-1}, \tilde{g}_{i,t}, \eta_t)$ ▷ 1-step-ahead weight prediction
12: Compute $\nabla f(\tilde{x}_{i,t}; z_{i,t})$, $z_{i,t} \sim \mathcal{D}_i$ ▷ Synchronization is finished in the next iteration
13: $g_t \leftarrow \frac{1}{n} \sum_{i \in [n]} g_{i,t}$
14: $x_t \leftarrow optimizer\,(x_{t-1}, g_t, \eta_t)$

3.2 Staleness Compensation

We introduce two staleness compensation methods in our distributed training pipeline: delay compensation and weight prediction. Note that in SAPipe, they are only applied to the first k layers of DNN, chosen by the partial staleness algorithm above, while the remaining layers are updated by normal distributed training procedure as shown in the blue part of Algorithm 1.

Delay compensation (DC). Inspired by DC-ASGD [34], we mitigate the effects of delayed gradients on the model with delay compensation. This method leverages Taylor expansion of the gradient function and efficient approximation of the Hessian matrix of the loss function. In the original design of DC-ASGD, the following delay-compensated gradient is used: $g_t \approx g_{t'} + \lambda g_{t'} \circ g_{t'} \circ (x_{t-1} - x_{t'-1})$, where $t > t'$, g_t is the gradient evaluated on x_{t-1}, and $\lambda > 0$ is a hyperparameter. Note that the delay compensation above is a diagonal approximation of the full matrix form: $g_{t'} \circ g_{t'} \circ (x_{t-1} - x_{t'-1}) \approx g_{t'} g_{t'}^{\top} (x_{t-1} - x_{t'-1})$. In this paper, we prefer to use the full-matrix form of DC to avoid the additional error caused by the diagonal approximation:

$$g_t \approx DC_{\lambda}(g_{t'}, x_{t-1} - x_{t'-1}) = g_{t'} + \lambda g_{t'} g_{t'}^{\top} (x_{t-1} - x_{t'-1}). \tag{1}$$

The detailed algorithm of SAPipe with DC is shown in Algorithm 2. We will show that by compensating the 1-step staleness, SAPipe-DC achieves a lower error bound than vanilla SAPipe, and converges as fast as the non-stale baseline.

Weight prediction (WP). In a 1-step staleness pipeline, model weights are always updated using the synchronized gradients in the last iteration, resulting in convergence problems with inconsistent weight. The goal of WP is to estimate the 1-step-ahead weights for forward pass and backward pass, and to obtain 1-step-ahead gradients, in order to counteract the 1-step staleness. We provide three options for weight prediction: 1) WP with local gradient in the current step, 2) WP with the latest synchronized gradient, and 3) WP with all the above combined and DC.

The algorithm of SAPipe with WP is given in Algorithm 3. Line 5 computes the normal gradients in the first step. In the following steps, Line 7 retrieves the local gradient cached in each worker, and Line 8 to Line 10 provide the options for weight prediction. We apply an extra optimizer update using predicted gradients (Line 11) in each worker to predict 1-step-ahead weights, and conduct forward and backward propagation to obtain the 1-step-ahead gradients (Line 12). Line 13 finishes the synchronization of the gradients from the last step, and Line 14 uses them to update the model. Note that the delayed gradients in the last step are computed based on 1-step predicted weights. Therefore, the 1-step staleness is compensated.

3.3 Optimized Runtime

Figure 1(b) shows the workflow of the staleness pipeline system. SAPipe's staleness pipeline incurs extra overhead, namely: (1) the computation of staleness compensation; (2) data transfer between the staleness buffer and other modules. To minimize the system overhead, we present an algorithm-system co-design. We fuse the computation operations in DC and WP methods and other small weight update functions into several batched kernels. This saves a significant amount of kernel

launching time. We further optimize the usage of staleness buffer. We use a double buffer system [28] to handle stale gradients, reducing the transferring overhead, and minimize the memory usage by sharing the staleness buffer between computation and communication. We also switch the location of the staleness buffer between CPU memory and GPU memory according to the execution time of communication and computation pipeline stages. This overlaps the transferring overhead of the staleness buffer between communication and computation devices.

4 Theoretical Results

We establish theoretical guarantees of the convergence of SAPipe for smooth but non-convex problems, using SGD as the optimizer: $optimizer_{SGD}(x_{t-1}, g_t, \eta_t) = x_{t-1} - \eta_t g_t$, with a constant learning rate $\eta_t = \eta$.

4.1 Assumptions

Assumption 1. *(Smoothness) We assume that $f(x; z)$, $\forall z$, are L-smooth: $\|\nabla f(x; z) - \nabla f(y; z)\| \leq L\|x - y\|, \forall x, y$, which implies $f(y; z) - f(x; z) \leq \langle \nabla f(x; z), y - x\rangle + \frac{L}{2}\|y - x\|^2$.*

Assumption 2. *For any stochastic gradient $g_{i,t} = \nabla f(x_{i,t-1}; z_{i,t})$, $z_{i,t} \sim \mathcal{D}_i$, where $\mathcal{D}_i$ is the local dataset on worker i, we assume bounded variance and ℓ_2-norm: $\mathbb{E}[\|g_{i,t} - \nabla F_i(x_{i,t-1})\|^2] \leq V_1$, $\|g_{i,t}\|^2 \leq V_2, \forall i \in [n], t \in [T]$. Furthermore, gradients from different workers are independent of each other.*

Assumption 3. *Introduced in [30], we assume bounded gradient diversity: $\frac{\sum_{i\in[n]} \|\nabla F_i(x)\|^2}{\|\sum_{i\in[n]} \nabla F_i(x)\|^2} \leq \rho, \forall x$.*

The gradient diversity shows to what extent the local gradients on different workers are distinguished from each other. It is easy to check that Assumption 3 implies the bounded difference between the local and global gradients: $\frac{1}{n}\sum_{i\in[n]} \|\nabla F_i(x) - \nabla F(x)\|^2 \leq \left(\frac{\rho}{n} - 1\right) \|\nabla F(x)\|^2$.

Assumption 4. *For DC in Eqn. (1), we assume that the objective function $f(x)$ is twice-differentiable. Thus, we have the Taylor's approximation on $x' \in \mathbb{R}^d$ with bounded remainder: $\|\nabla f(x) - (\nabla f(x') + \nabla^2 f(x')(x - x'))\| \leq M\|x - x'\|^2$, where $\nabla^2 f(x')$ is the Hessian matrix evaluated on x'. We also assume that the Hessian approximation error is upper-bounded by Δ, $\forall x \in \mathbb{R}^d$, i.e., $\|\nabla f(x)(\nabla f(x))^\top - \nabla^2 f(x)\|_2 \leq \Delta$, with vector-induced matrix norm $\|\cdot\|_2$.*

Note that the original paper of DC-ASGD proves that $\Delta \to 0$ when $t \to \infty$ under certain assumptions. Thus, it is reasonable to assume a bounded approximation error.

Assumption 5. *For simplicity, we assume that for partial staleness, the elements of every gradient are randomly and independently chosen to have 1-step staleness.*

Assumption 6. *There exists at least one global minimum x_*, where $F(x_*) \leq F(x), \forall x$. And we define the initial gap as $R_0 = F(x_0) - F(x_*)$.*

4.2 Convergence Analysis

We derive the following error bounds on the convergence of SAPipe under the above assumptions. All proofs can be found in Appendix A.

Theorem 2. *Under Assumptions 1, 2, 5 and 6, taking $\eta \leq \frac{1}{L}$, after T iterations, for vanilla SAPipe without DC or WP, we have the following error bound: $\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \leq \frac{2R_0}{\eta T} + \kappa Err_0 + Var_0$, where $Err_0 = L^2\eta^2 V_2$ and $Var_0 = \frac{L\eta V_1}{n}$.*

Remark 1. *Theorem 2 shows that the overall convergence error bound of vanilla SAPipe includes **2 main components: the gradient estimation error** Err_0 **and the variance error** Var_0. Err_0 is rooted in the 1-step staleness. Var_0 is incurred by the random sampling in stochastic gradients.*

Remark 2. *When no layer is updated with staleness, SAPipe is reduced to vanilla distributed SGD. In this case, the gradient estimation error κErr_0 vanishes with $\kappa = 0$, which results in the exact error bound of vanilla distributed SGD. On the contrary, when all the layers are updated with staleness, SAPipe is reduced to PipeSGD with the gradient estimation error Err_0 and $\kappa = 1$.*

Theorem 3. *Under Assumptions 1, 2, 4, 5 and 6, taking a small enough λ so that $\lambda\eta V_2 \leq 1 - \frac{1}{\sqrt{2}}$, and $\eta \leq \frac{1}{L}$, after T iterations, for Algorithm 2 (SAPipe-DC), we have the following error bound:*

$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \leq \frac{2R_0}{\eta T} + \kappa Err_{DC} + (1+\kappa)Var_0$, *where* $Err_{DC} = 128\eta^4 M^2 V_2^2 + 32\eta^2(1-\lambda)^2 L^2 V_2 + 32\eta^2\lambda^2\Delta^2 V_2 + 16\eta^3\lambda^2 L V_2^3$.

Remark 3. *Regardless of the variance error, the main difference between the error bounds of vanilla SAPipe and SAPipe-DC is the gradient estimation error. Note that if we have a small enough Hessian estimation error yielding very small M and Δ, by taking small enough η and making $\lambda \to 1$, the overall gradient estimation error of SAPipe-DC is smaller than that of PipeSGD, i.e., $Err_{DC} \leq Err_0$ under certain conditions. In other words,* ***SAPipe-DC has better convergence compared to PipeSGD when the Hessian approximation error is small enough****.*

Theorem 4. *Under Assumptions 1, 2, 3, 5 and 6, taking $\eta \leq \frac{1}{L}$, after T iterations, for Algorithm 3 (SAPipe-WP) with Option 1, we have the following error bound: $\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \leq \frac{2R_0}{\eta T} + \kappa Err_{WP-1} + Var_0$, where $Err_{WP-1} = L^2\eta^2\left[\left(\frac{\rho}{n}-1\right)V_2 + \left(3-\frac{\rho}{n}\right)V_1\right]$.*

Remark 4. *The main difference between Theorem 2 and Theorem 4 is the gradient estimation error, i.e., V_2 versus $\left(\frac{\rho}{n}-1\right)V_2 + \left(3-\frac{\rho}{n}\right)V_1$. When V_1 and ρ are small enough (i.e., $V_1 \leq \frac{2-\rho/n}{3-\rho/n}V_2$), SAPipe-WP with Option 1 produces a smaller error compared to the vanilla SAPipe, i.e., $Err_{WP-1} \leq Err_0$ under certain conditions. In other words,* ***SAPipe-WP with Option 1 has better convergence compared to vanilla SAPipe when the difference between the local datasets on different workers is small enough****.*

Theorem 5. *Under Assumptions 1, 2, 5 and 6, taking $\eta \leq \frac{1}{L}$, after T iterations, for Algorithm 3 (SAPipe-WP) with Option 2, we have the following error bound: $\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \leq \frac{2R_0}{\eta T} + \kappa Err_{WP-2} + Var_0$, where $Err_{WP-2} = \frac{L^2\eta^2}{1-2L^2\eta^2}(2V_1 + 2L^2\eta^2 V_2)$.*

Remark 5. *The main difference between Theorem 2 and Theorem 5 is the gradient estimation error, i.e., V_2 versus $\frac{1}{1-2L^2\eta^2}(2V_1+2L^2\eta^2V_2)$. When L is small enough (i.e., $L^2 \leq \frac{1-2V_1/V_2}{4\eta^2}$), SAPipe-WP with Option 2 produces a smaller error compared to vanilla SAPipe, i.e., $Err_{WP-2} \leq Err_0$ under certain conditions. In other words,* ***SAPipe-WP with Option 2 has better convergence compared to vanilla SAPipe when the objective function is "smooth" enough****.*

Theorem 6. *Under Assumptions 1, 2, 4, 5 and 6, taking $\eta \leq \frac{1}{L}$, after T iterations, for Algorithm 3 (SAPipe-WP) with Option 3, we have the following error bound: $\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \leq \frac{2R_0}{\eta T} + \kappa Err_{WP-3} + Var_0$, where $Err_{WP-3} = \eta^2 L^2[8\frac{V_1}{n} + 4\eta^4 M^2 V_2^2 + 2\eta^2 V_2(L^2(1-\lambda)^2 + \lambda^2\Delta^2)]$.*

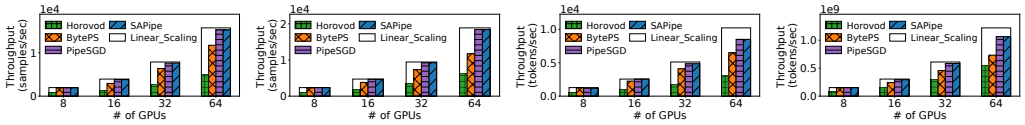
Remark 6. *The main difference between Theorem 2 and Theorem 6 is the gradient estimation error, i.e., V_2 versus $[8\frac{V_1}{n} + 4\eta^4 M^2 V_2^2 + 2\eta^2 V_2(L^2(1-\lambda)^2 + \lambda^2\Delta^2)]$. When η, M, Δ are small enough and n is large enough,* ***SAPipe-WP with Option 3 produces a smaller error compared to vanilla SAPipe, i.e.,*** *$Err_{WP-3} \leq Err_0$* ***under certain conditions****. Furthermore, λ provides a trade-off between L^2 and Δ^2. If the objective function is relatively "smooth" yielding $L < \Delta$, then $\lambda \to 0$ is preferred, and otherwise $\lambda \to 1$ is preferred. In practice, since both L and Δ are unknown and depend on the model architecture and datasets, tuning $\lambda \in [0,1]$ is required for better performance.*

Corollary 1. *For vanilla SAPipe, SAPipe-DC and SAPipe-WP with all 3 options, taking $\eta \propto \min\left(\frac{1}{\sqrt{T}}, \frac{1}{L}\right)$, we have $\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \leq \mathcal{O}\left(\frac{1}{\sqrt{T}}\right) + \mathcal{O}\left(\frac{V_1}{n\sqrt{T}}\right)$, same as vanilla SGD.*

Remark 7. *The corollary above shows that vanilla SAPipe, SAPipe-DC and SAPipe-WP with all 3 options converge to a critical point where $\|\nabla F(x_{t-1})\| \to 0$, when $T \to +\infty$, and the error decreases when there are more workers. In general, vanilla SAPipe, SAPipe-DC and SAPipe-WP have the same overall convergence rate $\mathcal{O}\left(\frac{1}{\sqrt{T}}\right)$. However, the detailed gradient estimation error varies for different algorithms, as shown in Theorems 2, 3, 4, 5, and 6. Under certain conditions such as low gradient variance, low gradient diversity, good smoothness, low Hessian approximation error, and specific choices of η and λ, one of these algorithms achieves the lowest convergence error. Hence,* ***hyperparameter tuning and algorithm selection are required in practice****.*

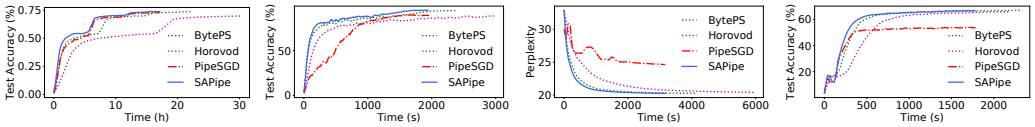
Table 2: Comparing model performance of SAPipe with baselines. We use perplexity (lower is better) as the metric for GPT-2, and accuracy (higher is better) for other models.

Model	VGG16		ResNet50		GPT-2	Transformer
Dataset	CIFAR-10	ImageNet	CIFAR-10	ImageNet	WikiText-2	Multi30K
BytePS	0.925 ± 0.0020	0.731 ± 0.0020	0.932 ± 0.0022	0.762 ± 0.0022	20.10 ± 0.23	0.673 ± 0.0033
PipeSGD	0.906 ± 0.0064	0.726 ± 0.0019	0.893 ± 0.0032	0.753 ± 0.0015	22.36 ± 0.33	0.531 ± 0.0044
SAPipe-DC	0.908 ± 0.0018	0.734 ± 0.0011	0.894 ± 0.0019	**0.758 ± 0.0150**	22.50 ± 0.48	0.526 ± 0.0031
SAPipe-WP-OPT1	**0.926 ± 0.0017**	0.718 ± 0.0020	**0.932 ± 0.0020**	0.757 ± 0.0022	21.80 ± 0.25	0.663 ± 0.0059
SAPipe-WP-OPT2	0.902 ± 0.0026	0.729 ± 0.0059	0.908 ± 0.0019	0.751 ± 0.0061	22.60 ± 0.09	0.662 ± 0.0007
SAPipe-WP-OPT3	0.914 ± 0.0020	**0.735 ± 0.0038**	0.897 ± 0.0034	**0.758 ± 0.0013**	**20.23 ± 0.32**	**0.668 ± 0.0032**
Over PipeSGD	2%	0.9%	3.9%	0.5%	2.13	13.7%



(a) VGG16 (b) ResNet50 (c) GPT-2 (d) Transformer

Figure 3: Training throughput.



(a) VGG16, ImageNet (b) ResNet50, CIFAR-10 (c) GPT-2, WikiText-2 (d) Transformer, Multi30K

Figure 4: Convergence of different models. x-axis is wall-clock training time, and y-axis is perplexity (lower is better) for GPT-2, and test accuracy (higher is better) for others.

5 Evaluation

5.1 Set-up

Testbed. We evaluate SAPipe [2] on 8 physical machines, each equipped with 90 CPU cores, 320GB memory, 8 Tesla V100 GPUs with NVLinks, and 100Gbps bandwidth between any two machines.

Models and datasets. We choose two CV models, VGG16 [27], ResNet50 [10], and two NLP models, pretrained GPT-2 [25], Transformer [29], as our benchmark models. The batch sizes per GPU are 128 images, 128 images, 80 tokens and 3200 tokens, respectively. We adopt SGD optimizer with 0.9 Polyak's momentum [24] and 5e-5 weight decay when training VGG16 and ResNet50 models, and Adam [14] optimizer with (0.9, 0.98) betas for NLP models. The global learning rates for VGG16, ResNet50 and GPT-2 are 0.1, 0.1, and 5e-5, respectively, and we follow the learning rate setting in [29] when training Transformer. SAPipe uses Option 3 in Algorithm 3 as the default staleness compensation method, with λ empirically set as 0.2.

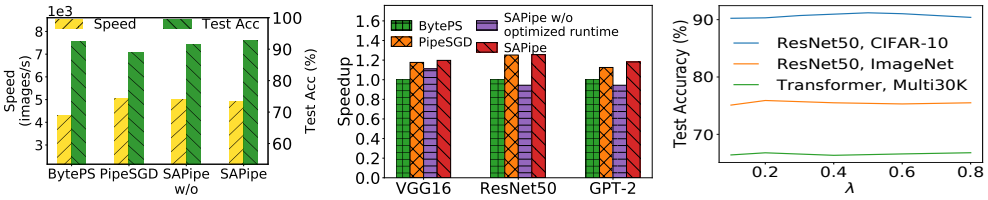
We train CV models on two datasets: (i) CIFAR-10 [16] and (ii) ImageNet [17]. We fine-tune the pretrained GPT-2 model on (iii) WikiText-2 language modeling dataset [20]. The Transformer model is trained on (iv) Multi30K [8] for WMT16 English-to-German Multimodal Translation task.

Baselines. We compare SAPipe with three state-of-the-art communication frameworks: (1) Horovod [26], a high-performance all-reduce paradigm; (2) BytePS [13], an optimized parameter-server architecture; (3) PipeSGD [19], a pipelined training framework with 1-step staleness. [3]. All baselines and SAPipe are run on PyTorch computation framework. [4]

[2] Code: https://github.com/ChenAris/sapipe.git

[3] Since PipeSGD is not open-sourced, we implement its staleness pipeline based on BytePS architecture.

[4] Magnified figures and additional experiment results are in the Appendix.



(a) VGG16 with partial staleness. (b) Runtime optimization. (c) Sensitivity analysis

Figure 5: Deep dive in SAPipe. "SAPipe w/o" denotes SAPipe without partial staleness.

5.2 Results and Analysis

Convergence. Figure 4 shows the convergence curves when training DNN models with 16 GPUs. We observe that SAPipe converges much faster than baselines. It not only achieves the highest training throughput, but also incurs very little influence on model convergence. For example, SAPipe is roughly 35%, 67% and 21% faster for VGG16 than BytePS, Horovod and PipeSGD, respectively. Though PipeSGD has comparable training speed, it converges much slower across models, and decreases final accuracy by 13% with Transformer. This reveals the obvious error incurred by 1-step staleness.

Staleness mitigation. Table 2 lists the final model performance of different frameworks. [5] SAPipe achieves comparable test accuracy and perplexity with non-stale baseline, BytePS. PipeSGD incurs significant performance drop on four models. With staleness compensation, SAPipe solves the convergence problem with staleness pipeline, while greatly boosting the training speed. Overall, SAPipe achieves significant improvement in accuracy/perplexity over PipeSGD of 2.6%, 3.5%, 13.7% and 2.13, for VGG16, ResNet50, Transformer and GPT-2, respectively.

We also compare different staleness compensation options of SAPipe with baselines. The best option varies among DNN training jobs. For example, "SAPipe-WP-OPT1" (*i.e.*, predicting weights using local gradients) achieves the highest test accuracy when training CV models on CIFAR-10, and "SAPipe-WP-OPT3" (*i.e.*, combined mitigation with WP and DC) is the best option for ImageNet, WikiText-2 and Multi30K datasets. Other options improve the convergence to some extent as compared to PipeSGD, but are suboptimal in these cases. The method to find the best staleness mitigation option for a given DNN task is our future work.

Scalability. Figure 3 shows the throughput of baselines and SAPipe when training with different numbers of workers. SAPipe achieves up to 57% and 203% speedups over BytePS and Horovod in all settings. Horovod does not apply communication scheduling, which leads to the worst throughput. BytePS enables preemptive communication scheduling, but it does not overlap gradient synchronization of the first few layers with computation. Thus, the communication overhead in BytePS is still large for GPT-2 model. PipeSGD performs the best among baselines, while SAPipe still achieves up to 6% speedup over PipeSGD, thanks to our high-performance runtime optimization.

Ablation study. We evaluate the effectiveness of each component of SAPipe. Figure 5(a) shows the throughput speedup and staleness mitigation with partial staleness when training VGG16 on CIFAR-10 dataset with 8 GPUs. We observe that with our partial staleness, SAPipe improves the final accuracy without slowing down training, by involving less delayed gradients.

We also compare SAPipe with BytePS, PipeSGD and SAPipe without runtime optimization when training DNN models with 32 GPUs, as shown in Figure 5(c). We use the speedup over the BytePS baseline as the throughput metric. "SAPipe w/o optimized execution" refers to a naive implementation of staleness-aware pipeline, which incurs non-negligible overhead for staleness mitigation. SAPipe achieves the highest throughput across all models, and the overhead of staleness mitigation decreases the training speed by 7% to 24% over SAPipe. Without runtime optimization, the staleness pipeline with mitigation methods could be slower than BytePS baseline by up to 6%.

Sensitivity. Figure 5(d) varies the value of hyperparameter λ in "SAPipe-WP-OPT3" method when training ResNet50 and Transformer. We observe that the staleness compensation method is not

[5] Horovod does not impact the convergence and has the same final accuracy as BytePS, so we omit it.

sensitive to the hyperparameter, and achieves the highest accuracy when it reaches 0.5, 0.2 and 0.2 when training on CIFAR-10, ImageNet and Multi30K datasets, respectively.

6 Related Work

Communication optimization. Many popular ML frameworks, such as TensorFlow [1] and PyTorch [22], enable overlapping communication with backward propagation by default. Recent works [12, 9, 23, 3] further overlap gradient synchronization with forward computation via tensor partitioning, at the cost of extra overhead. ASP [6] advocates asynchronous training to improve communication efficiency; without controlling the staleness, it leads to unstable convergence. SSP [11] is another asynchronous training protocol with bounded staleness. PipeSGD [19] restricts the staleness to one step in training. However, these methods do not mitigate the extra error caused by staleness, which degrades the model performance. We focus on synchronous pipelined training, without gradient compression. We minimize the number of stale layers, and propose multiple staleness compensation methods, achieving both high training throughput and comparable accuracy.

Staleness mitigation. Staleness mitigation is important for asynchronous SGD. Widely used approaches include staleness-aware rescaling of learning rate [33] and gradients [4], and delay compensation [34]. [15, 5] use linear weight prediction to narrow down the difference between models used in forward and backward passes for model parallelism. In our experiments, we find that simply using delay compensation or linear weight prediction in PipeSGD results in poor convergence, which calls for better strategies. Furthermore, the additional overhead incurred by staleness mitigation motivates our co-design of training algorithms and system optimization. However, the best staleness mitigation method varies in different DNN tasks, and this is the limitation of SAPipe that requires empirical experiments to find the optimal solution.

7 Conclusion

We present SAPipe, a performant and staleness-aware pipelined system to accelerate distributed DNN training without model performance loss. To fully overlap gradient synchronization communication with computation with minimal staleness, we introduce partial staleness, which restricts the number of layers learned with stale gradients. To further mitigate convergence issues caused by staleness, SAPipe adopts weight prediction and delay compensation. With an algorithm-system co-design, SAPipe achieves both better error bounds in theory, and high-performance runtime in practice.

Acknowledgments and Disclosure of Funding

We are thankful to the anonymous NeurIPS reviewers for their constructive feedback. This work was supported in part by Hong Kong Innovation and Technology Commission's Innovation and Technology Fund (Partnership Research Programme with ByteDance Limited, Award No. PRP/082/20FX), and grants from Hong Kong RGC under the contracts HKU 17204619, 17208920 and 17207621.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[2] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *NeurIPS*, 2017.

[3] Yixin Bao, Yanghua Peng, Yangrui Chen, and Chuan Wu. Preemptive All-reduce Scheduling for Expediting Distributed DNN Training. In *INFOCOM*, 2020.

[4] Saar Barkai, Ido Hakimi, and Assaf Schuster. Gap Aware Mitigation of Gradient Staleness. In *ICLR*, 2019.

[5] Chi-Chung Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. Efficient and Robust Parallel DNN Training through Model Parallelism on Multi-GPU Platform. *arXiv preprint arXiv:1809.02839*, 2018.

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large Scale Distributed Deep Networks. In *NeurIPS*, 2012.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 2019.

[8] Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. Multi30k: Multilingual english-german image descriptions. In *VL@ ACL*, 2016.

[9] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy Campbell. TicTac: Accelerating Distributed Deep Learning with Communication Scheduling. In *MLSys*, 2019.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

[11] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ML via a Stale Synchronous Parallel parameter server. In *NeurIPS*, 2013.

[12] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based Parameter Propagation for Distributed DNN Training . In *MLSys*, 2019.

[13] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *OSDI*, 2020.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[15] Atli Kosson, Vitaliy Chiley, Abhinav Venigalla, Joel Hestness, and Urs Koster. Pipelined Backpropagation at Scale: Training Large Models without Batches . In *MLSys*, 2021.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. 2009.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*, 2012.

[18] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling Distributed Machine Learning with the Parameter Server. In *OSDI*, 2014.

[19] Youjie Li, Mingchao Yu, Songze Li, Salman Avestimehr, Nam Sung Kim, and Alexander Schwing. Pipe-SGD: A Decentralized Pipelined SGD Framework for Distributed Deep Net Training. In *NeurIPS*, 2018.

[20] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. *arXiv preprint arXiv:1609.07843*, 2016.

[21] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed Precision Training. In *ICLR*, 2018.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library . In *NeurIPS*, 2019.

[23] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed DNN training acceleration. In *SOSP*, 2019.

[24] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[25] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 1(8):9, 2019.

[26] Alexander Sergeev and Mike Del Balso. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[27] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-scale Image Recognition. In *ICLR*, 2015.

[28] Xu Tan, Xiao-Wei Shen, Xiao-Chun Ye, Da Wang, Dong-Rui Fan, Lunkai Zhang, Wen-Ming Li, Zhi-Min Zhang, and Zhi-Min Tang. A non-stop double buffering mechanism for dataflow architecture. *JCST*, 2018.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *NeurIPS*, 2017.

[30] Dong Yin, Ashwin Pananjady, Maximilian Lam, Dimitris Papailiopoulos, Kannan Ramchandran, and Peter L. Bartlett. Gradient Diversity: a Key Ingredient for Scalable Distributed Learning. In *AISTATS*, 2018.

[31] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters. In *ATC*, 2017.

[32] Lingqi Zhang, Mohamed Wahib, and Satoshi Matsuoka. Understanding the overheads of launching CUDA kernels. In *ICPP*, 2019.

[33] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. Staleness-Aware Async-SGD for Distributed Deep Learning. In *IJCAI*, 2015.

[34] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhiming Ma, and Tie-Yan Liu. Asynchronous Stochastic Gradient Descent with Delay Compensation. In *ICML*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In Section 5 (Experiment) and Section 6 (related work)
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Section 4.1
   (b) Did you include complete proofs of all theoretical results? [Yes] Appendix A
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Section 4.1 and source code
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Publicly available benchmarks

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Proofs

In this section we provide the detailed proofs of the theoretical analysis.

Lemma 1. *(General error bound) Under Assumption 1, taking $\eta \leq \frac{1}{L}$, in the tth step, for the distributed SGD with the general gradient estimator $g_{i,t}$ (the gradient estimator produced by the ith worker in the tth step), we have the following error bound in expectation conditional on $x_{t'}$ where $t' < t$:*

$$\begin{aligned}
&\mathbb{E}[F(x_t)] \\
&\leq F(x_{t-1}) - \frac{\eta}{2}\|\nabla F(x_{t-1})\|^2 \\
&\quad + \frac{\eta}{2}\left\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2 + \frac{L\eta^2}{2}\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right].
\end{aligned}$$

The theorem states that in each iteration, the loss function $F(x)$ is expected to decrease by $\frac{\eta}{2}\|\nabla F(x_{t-1})\|^2$, with the error caused by the variance of the stochastic gradient estimator $\frac{L\eta^2}{2}\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right]$ (i.e., variance) and the error caused by the difference between the true gradient and the gradient estimator $\frac{\eta}{2}\left\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2$ (i.e., bias).

Proof. In the tth step, using L-smoothness, we have

$$\begin{aligned}
&F(x_t) \\
&\leq F(x_{t-1}) + \langle \nabla F(x_{t-1}), x_t - x_{t-1}\rangle + \frac{L}{2}\|x_t - x_{t-1}\|^2 \\
&= F(x_{t-1}) + \left\langle \nabla F(x_{t-1}), -\eta\frac{1}{n}\sum_{i\in[n]} g_{i,t}\right\rangle + \frac{L}{2}\left\|-\eta\frac{1}{n}\sum_{i\in[n]} g_{i,t}\right\|^2.
\end{aligned}$$

Taking expectation on both sides conditional on $x_{t'}$ where $t' < t$, we have

$$\mathbb{E}[F(x_t)] \leq F(x_{t-1}) - \eta\underbrace{\left\langle \nabla F(x_{t-1}), \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\rangle}_{①} + \frac{L\eta^2}{2}\underbrace{\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t}\right\|^2\right]}_{②}.$$

We bound the terms step by step.

First, for ①, we have

$$\begin{aligned}
&① \\
&= \left\langle \nabla F(x_{t-1}), \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\rangle \\
&= -\frac{1}{2}\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\|^2 + \frac{1}{2}\|\nabla F(x_{t-1})\|^2 + \frac{1}{2}\left\|\frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2.
\end{aligned}$$

For ②, we have

②

$$
\begin{aligned}
&= \mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t}\right\|^2\right] \\
&= \mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}] + \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right] \\
&= \mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right] + \left\|\frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2.
\end{aligned}
$$

Combining the ingredients above together, we have

$$
\begin{aligned}
&\mathbb{E}[F(x_t)] \\
&\le F(x_{t-1}) - \eta \underbrace{\left\langle \nabla F(x_{t-1}), \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\rangle}_{①} + \frac{L\eta^2}{2}\underbrace{\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t}\right\|^2\right]}_{②} \\
&\le F(x_{t-1}) - \eta\left[-\frac{1}{2}\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\|^2 + \frac{1}{2}\|\nabla F(x_{t-1})\|^2 + \frac{1}{2}\left\|\frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right] \\
&\quad + \frac{L\eta^2}{2}\left[\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right] + \left\|\frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right] \\
&= F(x_{t-1}) + \frac{\eta}{2}\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\|^2 - \frac{\eta}{2}\|\nabla F(x_{t-1})\|^2 - \frac{\eta}{2}\left\|\frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2 \\
&\quad + \frac{L\eta^2}{2}\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right] + \frac{L\eta^2}{2}\left\|\frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2 \\
&\le F(x_{t-1}) - \frac{\eta}{2}\|\nabla F(x_{t-1})\|^2 \\
&\quad + \frac{\eta}{2}\left\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2 + \frac{L\eta^2}{2}\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right]. \qquad \triangleright\ \eta \le \tfrac{1}{L}
\end{aligned}
$$

□

Using Lemma 1, we establish the error bound of the convergence of vanilla SAPipe, Algorithm 2 (SAPipe-DC) Algorithm 3 (SAPipe-WP).

Theorem 2. *Under Assumption 1, 2, 5, and 6, taking $\eta \le \frac{1}{L}$, after T iterations, for vanilla SAPipe without DC or WP, we have the following error bound:*

$$
\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \le \frac{2R_0}{\eta T} + \kappa Err_0 + Var_0,
$$

where $Err_0 = L^2\eta^2 V_2$, $Var_0 = \frac{L\eta V_1}{n}$.

Proof. For vanilla SAPipe, we have $\mathbb{E}[g_{i,t}] = \nabla F_i(x_{t-2})$ if the gradient has one-step staleness.

Thus, for the variance term, we have

$$
\begin{aligned}
&\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right] \\
&= \frac{1}{n^2}\sum_{i\in[n]}\mathbb{E}\left[\|g_{i,t}-\mathbb{E}[g_{i,t}]\|^2\right] \\
&= \frac{1}{n^2}\sum_{i\in[n]}\mathbb{E}\left[\|g_{i,t}-\nabla f_i(x_{t-2})\|^2\right] \\
&\le \frac{V_1}{n}.
\end{aligned}
$$

For the error of the gradient estimation, we have

$$
\begin{aligned}
&\left\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2 \\
&= \left\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\nabla F_i(x_{t-2})\right\|^2 \\
&= \|\nabla F(x_{t-1}) - \nabla F(x_{t-2})\|^2 \\
&\le L^2\|x_{t-1}-x_{t-2}\|^2 && \triangleright\ L\text{-smoothness} \\
&= L^2\left\|\eta\frac{1}{n}\sum_{i\in[n]} g_{i,t-1}\right\|^2 \\
&= L^2\eta^2\left\|\frac{1}{n}\sum_{i\in[n]}\nabla f_i(x_{t-3})\right\|^2.
\end{aligned}
$$

Putting all the ingredients together, we have

$$
\begin{aligned}
&\mathbb{E}[F(x_t)] \\
&\le F(x_{t-1}) - \frac{\eta}{2}\|\nabla F(x_{t-1})\|^2 \\
&\quad + \frac{\eta}{2}\left\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2 + \frac{L\eta^2}{2}\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right] \\
&\le F(x_{t-1}) - \frac{\eta}{2}\|\nabla F(x_{t-1})\|^2 + \frac{\kappa\eta}{2}L^2\eta^2\left\|\frac{1}{n}\sum_{i\in[n]}\nabla f_i(x_{t-3})\right\|^2 + \frac{L\eta^2}{2}\frac{V_1}{n}.
\end{aligned}
$$

By re-arranging the terms, we have

$$
\|\nabla F(x_{t-1})\|^2 \le \frac{2\mathbb{E}[F(x_{t-1})-F(x_t)]}{\eta} + \kappa L^2\eta^2\left\|\frac{1}{n}\sum_{i\in[n]}\nabla f_i(x_{t-3})\right\|^2 + \frac{L\eta V_1}{n}.
$$

By telescoping and taking total expectation, after T iterations, we have

$$
\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right]
$$

$$\leq \frac{2\mathbb{E}[F(x_0) - F(x_*)]}{\eta T} + L^2\eta^2 \frac{1}{T}\sum_{t=3}^{T} \mathbb{E}\left\| \frac{1}{n}\sum_{i\in[n]} \nabla f_i(x_{t-3}) \right\|^2 + \frac{L\eta V_1}{n}$$

$$\leq \frac{2\mathbb{E}[F(x_0) - F(x_*)]}{\eta T} + \kappa L^2\eta^2 V_2 + \frac{L\eta V_1}{n}. \qquad \triangleright \text{ Assumption 2}$$

□

Theorem 3. *Under Assumption 1, 2, 4, 5, and 6, taking a small enough λ so that $\lambda\eta V_2 \leq 1 - \frac{1}{\sqrt{2}}$, and $\eta \leq \frac{1}{L}$, after T iterations, for Algorithm 2 (SAPipe-DC), we have the following error bound:*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \leq \frac{2R_0}{\eta T} + \kappa Err_{DC} + (1+\kappa)Var_0,$$

where $Err_{DC} = 128\eta^4 M^2 V_2^2 + 32\eta^2(1-\lambda)^2 L^2 V_2 + 32\eta^2\lambda^2\Delta^2 V_2 + 16\eta^3\lambda^2 L V_2^3$.

Proof. Thus, for the variance term, we have

$$\begin{aligned}
&\mathbb{E}\left[\left\|g_t^{DC} - \mathbb{E}[g_t^{DC}]\right\|^2\right] \\
&= \mathbb{E}\left[\left\|g_t + \lambda g_t g_t^\top (x_{t-1} - x_{t-2}) - \mathbb{E}[g_t + \lambda g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2\right] \\
&= \mathbb{E}\left[\left\|g_t - \mathbb{E}[g_t] + \lambda g_t g_t^\top (x_{t-1} - x_{t-2}) - \mathbb{E}[\lambda g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2\right] \\
&\leq 2\mathbb{E}\left[\|g_t - \mathbb{E}[g_t]\|^2\right] + 2\mathbb{E}\left[\left\|\lambda g_t g_t^\top (x_{t-1} - x_{t-2}) - \mathbb{E}[\lambda g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2\right] \\
&\leq \frac{2V_1}{n} + 2\mathbb{E}\left[\left\|\lambda g_t g_t^\top (x_{t-1} - x_{t-2}) - \mathbb{E}[\lambda g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2\right] \\
&\leq \frac{2V_1}{n} + 2\lambda^2\mathbb{E}\left[\left\|g_t g_t^\top (x_{t-1} - x_{t-2})\right\|^2\right] \\
&\leq \frac{2V_1}{n} + 2\lambda^2\mathbb{E}\left[\|g_t\|^2\|g_t\|^2\|x_{t-1} - x_{t-2}\|^2\right] \\
&\leq \frac{2V_1}{n} + 2\lambda^2 V_2^2\|x_{t-1} - x_{t-2}\|^2.
\end{aligned}$$

For the error of the gradient estimation, we have

$$\begin{aligned}
&\left\|\nabla F(x_{t-1}) - \mathbb{E}[g_t^{DC}]\right\|^2 \\
&= \left\|\nabla F(x_{t-1}) - \mathbb{E}[g_t + \lambda g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2 \\
&= \left\|\nabla F(x_{t-1}) - \mathbb{E}[g_t + \nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2})] + \mathbb{E}[g_t + \nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2})] - \mathbb{E}[g_t + \lambda g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2 \\
&\leq 2\left\|\nabla F(x_{t-1}) - \mathbb{E}[g_t + \nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2})]\right\|^2 \\
&\quad + 2\left\|\mathbb{E}[g_t + \nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2})] - \mathbb{E}[g_t + \lambda g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2 \\
&\leq 2M^2\|x_{t-1} - x_{t-2}\|^4 + 2\left\|\mathbb{E}[\nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2})] - \mathbb{E}[\lambda g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2 \\
&\qquad \triangleright \text{ Assumption 4} \\
&\leq 2M^2\|x_{t-1} - x_{t-2}\|^4 \\
&\quad + 2\left\|\mathbb{E}[(1-\lambda)\nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2}) + \lambda(\nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2}) - g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2 \\
&\leq 2M^2\|x_{t-1} - x_{t-2}\|^4 + 4(1-\lambda)^2\left\|\mathbb{E}[\nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2})]\right\|^2 \\
&\quad + 4\lambda^2\left\|\mathbb{E}[\nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2}) - g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2 \\
&\leq 2M^2\|x_{t-1} - x_{t-2}\|^4 + 4(1-\lambda)^2 L^2\left\|x_{t-1} - x_{t-2}\right\|^2 \\
&\quad + 4\lambda^2\left\|\mathbb{E}[\nabla^2 f(x_{t-2})(x_{t-1} - x_{t-2}) - g_t g_t^\top (x_{t-1} - x_{t-2})]\right\|^2 \\
&\leq 2M^2\|x_{t-1} - x_{t-2}\|^4 + 4(1-\lambda)^2 L^2\|x_{t-1} - x_{t-2}\|^2 + 4\lambda^2\Delta^2\|x_{t-1} - x_{t-2}\|^2. \\
&\qquad \triangleright \text{ Assumption 4}
\end{aligned}$$

To finish the proof, we need to establish the upper bound of $\|x_{t-1} - x_{t-2}\|$. It is easy to check that the term $a_t = \|x_t - x_{t-1}\|$ has the following recursive inequality:

$$\begin{aligned}
&\|x_t - x_{t-1}\| \\
&= \|\eta(g_t + \lambda g_t g_t^\top (x_{t-1} - x_{t-2}))\| \\
&\leq \eta\|g_t\| + \eta\lambda\|g_t g_t^\top (x_{t-1} - x_{t-2})\| && \triangleright \text{ triangle inequality} \\
&\leq \eta\|g_t\| + \eta\lambda\|g_t\|^2\|x_{t-1} - x_{t-2}\| \\
&\leq \eta\sqrt{V_2} + \eta\lambda V_2\|x_{t-1} - x_{t-2}\|.
\end{aligned}$$

Or, $a_t \leq \eta\sqrt{V_2} + \eta\lambda V_2 a_{t-1}$. Note that $a_1 = \|x_1 - x_0\| = \|\eta g_1\| \leq \eta\sqrt{V_2}$. Thus, we have $\forall t \geq 2$:

$$\|x_t - x_{t-1}\| \leq \eta\sqrt{V_2}\sum_{\tau=0}^{t-2}(\lambda\eta V_2)^\tau + \eta\sqrt{V_2}(\lambda\eta V_2)^{t-1} \leq \frac{2\eta\sqrt{V_2}}{1-\lambda\eta V_2} \leq 2\sqrt{2}\eta\sqrt{V_2}.$$

Putting all the ingredients together, we have

$$\begin{aligned}
&\mathbb{E}[F(x_t)] \\
&\leq F(x_{t-1}) - \frac{\eta}{2}\|\nabla F(x_{t-1})\|^2 \\
&\quad + \frac{\kappa\eta}{2}[2M^2\|x_{t-1} - x_{t-2}\|^4 + 4(1-\lambda)^2L^2\|x_{t-1} - x_{t-2}\|^2 + 4\lambda^2\Delta^2\|x_{t-1} - x_{t-2}\|^2] \\
&\quad + \frac{\kappa L\eta^2}{2}[\frac{2V_1}{n} + 2\lambda^2V_2^2\|x_{t-1} - x_{t-2}\|^2] + \frac{(1-\kappa)L\eta^2V_1}{2n} \\
&\leq F(x_{t-1}) - \frac{\eta}{2}\|\nabla F(x_{t-1})\|^2 \\
&\quad + \kappa(64\eta^5M^2V_2^2 + 16\eta^3(1-\lambda)^2L^2V_2 + 16\eta^3\lambda^2\Delta^2V_2 + 8\eta^4\lambda^2LV_2^3) + \frac{(1+\kappa)\eta^2LV_1}{2n}.
\end{aligned}$$

By re-arranging the terms, we have

$$\begin{aligned}
&\|\nabla F(x_{t-1})\|^2 \\
&\quad + \kappa(128\eta^4M^2V_2^2 + 32\eta^2(1-\lambda)^2L^2V_2 + 32\eta^2\lambda^2\Delta^2V_2 + 16\eta^3\lambda^2LV_2^3) + \frac{(1+\kappa)\eta LV_1}{n}.
\end{aligned}$$

By telescoping and taking total expectation, after T iterations, we have

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \\
&\leq \frac{2\mathbb{E}[F(x_0) - F(x_*)]}{\eta T} \\
&\quad + \kappa(128\eta^4M^2V_2^2 + 32\eta^2(1-\lambda)^2L^2V_2 + 32\eta^2\lambda^2\Delta^2V_2 + 16\eta^3\lambda^2LV_2^3) + \frac{(1+\kappa)\eta LV_1}{n}.
\end{aligned}$$

□

Theorem 4. *Under Assumption 1, 2, 3, 5, and 6, taking $\eta \leq \frac{1}{L}$, after T iterations, for Algorithm 3 (SAPipe-WP) with Option 1, we have the following error bound:*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \leq \frac{2R_0}{\eta T} + \kappa Err_{WP-1} + Var_0,$$

where $Err_{WP-1} = L^2\eta^2\left[\left(\frac{\rho}{n} - 1\right)V_2 + \left(3 - \frac{\rho}{n}\right)V_1\right]$.

Proof. For SAPipe-WP with Option 1, we have $\mathbb{E}[g_{i,t}] = \nabla F_i(\tilde{x}_{i,t-1})$.

Thus, for the variance term, since $g_{i,t}$ is a valid gradient, similar to Theorem 2, we have

$$\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]}g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right]$$

$$
\begin{aligned}
&= \frac{1}{n^2} \sum_{i \in [n]} \mathbb{E}\left[\|g_{i,t} - \mathbb{E}[g_{i,t}]\|^2 \right] \\
&\leq \frac{V_1}{n}.
\end{aligned}
$$

For the error of the gradient estimation, we have

$$
\begin{aligned}
&\left\| \nabla F(x_{t-1}) - \frac{1}{n} \sum_{i \in [n]} \mathbb{E}[g_{i,t}] \right\|^2 \\
&= \left\| \nabla F(x_{t-1}) - \frac{1}{n} \sum_{i \in [n]} \nabla F_i(\tilde{x}_{i,t-1}) \right\|^2 \\
&= \left\| \frac{1}{n} \sum_{i \in [n]} \nabla F_i(x_{t-1}) - \frac{1}{n} \sum_{i \in [n]} \nabla F_i(\tilde{x}_{i,t-1}) \right\|^2 \\
&\leq \frac{1}{n} \sum_{i \in [n]} \|\nabla F_i(x_{t-1}) - \nabla F_i(\tilde{x}_{i,t-1})\|^2 \\
&\leq L^2 \frac{1}{n} \sum_{i \in [n]} \|x_{t-1} - \tilde{x}_{i,t-1}\|^2 && \triangleright\ L\text{-smoothness} \\
&= L^2 \frac{1}{n} \sum_{i \in [n]} \left\| \eta \frac{1}{n} \sum_{j \in [n]} (g_{j,t-1}) - \eta g_{i,t-1} \right\|^2 \\
&= \frac{L^2 \eta^2}{n} \sum_{i \in [n]} \left\| \frac{1}{n} \sum_{j \in [n]} (g_{j,t-1}) - g_{i,t-1} \right\|^2
\end{aligned}
$$

Putting all the ingredients together, we have

$$
\begin{aligned}
&\mathbb{E}[F(x_t)] \\
&\leq F(x_{t-1}) - \frac{\eta}{2} \|\nabla F(x_{t-1})\|^2 \\
&\quad + \frac{\eta}{2} \left\| \nabla F(x_{t-1}) - \frac{1}{n} \sum_{i \in [n]} \mathbb{E}[g_{i,t}] \right\|^2 + \frac{L\eta^2}{2} \mathbb{E}\left[\left\| \frac{1}{n} \sum_{i \in [n]} g_{i,t} - \frac{1}{n} \sum_{i \in [n]} \mathbb{E}[g_{i,t}] \right\|^2 \right] \\
&\leq F(x_{t-1}) - \frac{\eta}{2} \|\nabla F(x_{t-1})\|^2 + \frac{\kappa \eta}{2} \frac{L^2 \eta^2}{n} \sum_{i \in [n]} \left\| \frac{1}{n} \sum_{j \in [n]} (g_{j,t-1}) - g_{i,t-1} \right\|^2 + \frac{L\eta^2}{2} \frac{V_1}{n}.
\end{aligned}
$$

By re-arranging the terms, we have

$$
\|\nabla F(x_{t-1})\|^2 \leq \frac{2\mathbb{E}[F(x_{t-1}) - F(x_t)]}{\eta} + \frac{\kappa L^2 \eta^2}{n} \sum_{i \in [n]} \left\| \frac{1}{n} \sum_{j \in [n]} (g_{j,t-1}) - g_{i,t-1} \right\|^2 + \frac{L\eta V_1}{n}.
$$

By telescoping and taking total expectation, after T iterations, we have

$$
\begin{aligned}
&\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\|\nabla F(x_{t-1})\|^2 \right] \\
&\leq \frac{2\mathbb{E}[F(x_0) - F(x_*)]}{\eta T} + \frac{1}{T} \sum_{t=3}^{T} L^2 \eta^2 \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \left\| \frac{1}{n} \sum_{j \in [n]} (g_{j,t-1}) - g_{i,t-1} \right\|^2 + \frac{L\eta V_1}{n}.
\end{aligned}
$$

To finish the proof, we establish the following upper bound for $\forall i, j \in [n], i \neq j$

$$
\begin{aligned}
&\frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\frac{1}{n}\sum_{j\in[n]}(g_{j,t-1}) - g_{i,t-1}\right\|^2 \\
&= \frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\left(\frac{1}{n}\sum_{j\in[n]}\nabla f_j(\tilde{x}_{t-2})\right) - \nabla f_i(\tilde{x}_{t-2})\right\|^2 \\
&= \frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\left(\frac{1}{n}\sum_{j\in[n]}\nabla f_j(\tilde{x}_{t-2})\right) - \nabla f_i(\tilde{x}_{t-2}) \pm (\nabla F(\tilde{x}_{t-2}) - \nabla F_i(\tilde{x}_{t-2}))\right\|^2 \\
&= \frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\left(\frac{1}{n}\sum_{j\in[n]}\nabla f_j(\tilde{x}_{t-2})\right) - \nabla F(\tilde{x}_{t-2})\right\|^2 \\
&\quad + \frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\nabla F_i(\tilde{x}_{t-2}) - \nabla f_i(\tilde{x}_{t-2})\right\|^2 \\
&\quad + \frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\nabla F(\tilde{x}_{t-2}) - \nabla F_i(\tilde{x}_{t-2})\right\|^2 \\
&\leq \frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\nabla F(\tilde{x}_{t-2}) - \nabla F_i(\tilde{x}_{t-2})\right\|^2 + 2V_1 \\
&\leq \left(\frac{\rho}{n} - 1\right)\mathbb{E}\left\|\nabla F(\tilde{x}_{t-2})\right\|^2 + 2V_1 && \triangleright \text{ Assumption 3} \\
&\leq \left(\frac{\rho}{n} - 1\right)V_1' + 2V_1 && \triangleright \text{ Assumption 2} \\
&= \left(\frac{\rho}{n} - 1\right)(V_2 - V_1) + 2V_1 && \triangleright \text{ Assumption 2, } V_2 = V_1 + V_1' \\
&\leq \left(\frac{\rho}{n} - 1\right)V_2 + \left(3 - \frac{\rho}{n}\right)V_1.
\end{aligned}
$$

Finally, putting all the ingredients above together, we have

$$
\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \\
&\leq \frac{2\mathbb{E}[F(x_0) - F(x_*)]}{\eta T} + \kappa L^2\eta^2\left[\left(\frac{\rho}{n} - 1\right)V_2 + \left(3 - \frac{\rho}{n}\right)V_1\right] + \frac{L\eta V_1}{n}.
\end{aligned}
$$

□

Theorem 5. *Under Assumption 1, 2, 5, and 6, taking $\eta \leq \frac{1}{L}$, after T iterations, for Algorithm 3 (SAPipe-WP) with Option 2, we have the following error bound:*

$$
\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \leq \frac{2R_0}{\eta T} + \kappa Err_{WP-2} + Var_0,
$$

where $Err_{WP-2} = \frac{L^2\eta^2}{1-2L^2\eta^2}(2V_1 + 2L^2\eta^2 V_2)$.

Proof. For SAPipe-WP with Option 2, we have $\mathbb{E}[g_{i,t}] = \nabla F_i(\tilde{x}_{t-1})$.

Thus, for the variance term, since $g_{i,t}$ is a valid gradient, similar to Theorem 2, we have

$$
\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]} g_{i,t} - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right]
$$

$$
\begin{aligned}
&= \frac{1}{n^2} \sum_{i \in [n]} \mathbb{E}\left[\|g_{i,t} - \mathbb{E}[g_{i,t}]\|^2 \right] \\
&\leq \frac{V_1}{n}.
\end{aligned}
$$

For the error of the gradient estimation, we have

$$
\begin{aligned}
&\left\| \nabla F(x_{t-1}) - \frac{1}{n} \sum_{i \in [n]} \mathbb{E}[g_{i,t}] \right\|^2 \\
&= \left\| \nabla F(x_{t-1}) - \frac{1}{n} \sum_{i \in [n]} \nabla F_i(\tilde{x}_{t-1}) \right\|^2 \\
&= \|\nabla F(x_{t-1}) - \nabla F(\tilde{x}_{t-1})\|^2 \\
&\leq L^2 \|x_{t-1} - \tilde{x}_{t-1}\|^2 && \triangleright\ L\text{-smoothness} \\
&= L^2 \left\| \eta \frac{1}{n} \sum_{i \in [n]} (g_{i,t-1} - g_{i,t-2}) \right\|^2 \\
&= L^2 \eta^2 \left\| \frac{1}{n} \sum_{i \in [n]} (\nabla f(\tilde{x}_{t-2}; z_{i,t-2}) - \nabla f(\tilde{x}_{t-3}; z_{i,t-3})) \right\|^2.
\end{aligned}
$$

Putting all the ingredients together, we have

$$
\begin{aligned}
&\mathbb{E}[F(x_t)] \\
&\leq F(x_{t-1}) - \frac{\eta}{2} \|\nabla F(x_{t-1})\|^2 \\
&\quad + \frac{\eta}{2} \left\| \nabla F(x_{t-1}) - \frac{1}{n} \sum_{i \in [n]} \mathbb{E}[g_{i,t}] \right\|^2 + \frac{L\eta^2}{2} \mathbb{E}\left[\left\| \frac{1}{n} \sum_{i \in [n]} g_{i,t} - \frac{1}{n} \sum_{i \in [n]} \mathbb{E}[g_{i,t}] \right\|^2 \right] \\
&\leq F(x_{t-1}) - \frac{\eta}{2} \|\nabla F(x_{t-1})\|^2 + \frac{\kappa \eta}{2} L^2 \eta^2 \left\| \frac{1}{n} \sum_{i \in [n]} (\nabla f(\tilde{x}_{t-2}; z_{i,t-2}) - \nabla f(\tilde{x}_{t-3}; z_{i,t-3})) \right\|^2 \\
&\quad + \frac{L\eta^2}{2} \frac{V_1}{n}.
\end{aligned}
$$

By re-arranging the terms, we have

$$
\begin{aligned}
&\|\nabla F(x_{t-1})\|^2 \\
&\leq \frac{2\mathbb{E}[F(x_{t-1}) - F(x_t)]}{\eta} + \kappa L^2 \eta^2 \left\| \frac{1}{n} \sum_{i \in [n]} (\nabla f(\tilde{x}_{t-2}; z_{i,t-2}) - \nabla f(\tilde{x}_{t-3}; z_{i,t-3})) \right\|^2 \\
&\quad + \frac{L\eta V_1}{n}.
\end{aligned}
$$

By telescoping and taking total expectation, after T iterations, we have

$$
\begin{aligned}
&\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\|\nabla F(x_{t-1})\|^2 \right] \\
&\leq \frac{2\mathbb{E}[F(x_0) - F(x_*)]}{\eta T} + \frac{1}{T} \sum_{t=3}^{T} \kappa L^2 \eta^2 \mathbb{E} \left\| \frac{1}{n} \sum_{i \in [n]} (\nabla f(\tilde{x}_{t-2}; z_{i,t-2}) - \nabla f(\tilde{x}_{t-3}; z_{i,t-3})) \right\|^2 \\
&\quad + \frac{L\eta V_1}{n}.
\end{aligned}
$$

To finish the proof, we establish the following upper bound

$$
\begin{aligned}
&\mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}(\nabla f(\tilde{x}_{t-2};z_{i,t-2})-\nabla f(\tilde{x}_{t-3};z_{i,t-3}))\right\|^2\\
&\leq \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}[(\nabla f(\tilde{x}_{t-2};z_{i,t-2})-\nabla F(\tilde{x}_{t-2}))+(\nabla F(\tilde{x}_{t-3})-\nabla f(\tilde{x}_{t-3};z_{i,t-3}))+(\nabla F(\tilde{x}_{t-2})-\nabla F(\tilde{x}_{t-3}))]\right\|^2\\
&\leq \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}(\nabla f(\tilde{x}_{t-2};z_{i,t-2})-\nabla F(\tilde{x}_{t-2}))\right\|^2+\mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}(\nabla F(\tilde{x}_{t-3})-\nabla f(\tilde{x}_{t-3};z_{i,t-3}))\right\|^2\\
&\quad+\mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}(\nabla F(\tilde{x}_{t-2})-\nabla F(\tilde{x}_{t-3}))\right\|^2\\
&\leq \mathbb{E}\left\|\nabla F(\tilde{x}_{t-2})-\nabla F(\tilde{x}_{t-3})\right\|^2+2V_1\\
&\leq L^2\mathbb{E}\left\|\tilde{x}_{t-2}-\tilde{x}_{t-3}\right\|^2+2V_1\\
&= L^2\mathbb{E}\left\|x_{t-3}-x_{t-4}-\eta(g_{t-3}-g_{t-4})\right\|^2+2V_1\\
&\leq 2L^2\mathbb{E}\left\|x_{t-3}-x_{t-4}\right\|^2+2L^2\eta^2\mathbb{E}\left\|g_{t-3}-g_{t-4}\right\|^2+2V_1\\
&\leq 2L^2\mathbb{E}\left\|\eta g_{t-3}\right\|^2+2L^2\eta^2\mathbb{E}\left\|g_{t-3}-g_{t-4}\right\|^2+2V_1\\
&\leq 2L^2\eta^2\mathbb{E}\left\|g_{t-3}-g_{t-4}\right\|^2+2V_1+2L^2\eta^2V_2,
\end{aligned}
$$

where $g_t=\frac{1}{n}\sum_{i\in[n]}g_{i,t}$.

Thus, we have the following recursive upper bound

$$\mathbb{E}\left\|g_{t-1}-g_{t-2}\right\|^2\leq 2L^2\eta^2\mathbb{E}\left\|g_{t-3}-g_{t-4}\right\|^2+2V_1+2L^2\eta^2V_2,$$

and at the very beginning, there is no staleness, thus $g_1-g_2=0$.

Thus, using $\eta\leq\frac{1}{2L}$, we have

$$\mathbb{E}\left\|g_{t-1}-g_{t-2}\right\|^2\leq\frac{1}{1-2L^2\eta^2}(2V_1+2L^2\eta^2V_2).$$

Finally, putting all the ingredients together, we have

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right]\leq\frac{2\mathbb{E}[F(x_0)-F(x_*)]}{\eta T}+\frac{\kappa L^2\eta^2}{1-2L^2\eta^2}(2V_1+2L^2\eta^2V_2)+\frac{L\eta V_1}{n}.$$

□

Theorem 6. *Under Assumption 1, 2, 4, 5, and 6, taking $\eta\leq\frac{1}{L}$, after T iterations, for Algorithm 3 (SAPipe-WP) with Option 3, we have the following error bound:*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right]\leq\frac{2R_0}{\eta T}+\kappa Err_{WP-3}+Var_0,$$

where $Err_{WP-3}=\eta^2L^2[8\frac{V_1}{n}+4\eta^4M^2V_2^2+2\eta^2V_2(L^2(1-\lambda)^2+\lambda^2\Delta^2)]$.

Proof. Note that similar to the other options, in this case we still have $g_{i,t}$ as a valid stochastic gradient. Thus, for the variance term, we have

$$
\begin{aligned}
&\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i\in[n]}g_{i,t}-\frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2\right]\\
&=\frac{1}{n^2}\sum_{i\in[n]}\mathbb{E}\left[\|g_{i,t}-\mathbb{E}[g_{i,t}]\|^2\right]\\
&\leq\frac{V_1}{n}.
\end{aligned}
$$

For the error of the gradient estimation, we have

$$
\begin{aligned}
&\left\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_{i,t}]\right\|^2 \\
&= \left\|\nabla F(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\nabla F_i(\tilde{x}_{i,t-1})\right\|^2 \\
&= \left\|\frac{1}{n}\sum_{i\in[n]}\nabla F_i(x_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\nabla F_i(\tilde{x}_{i,t-1})\right\|^2 \\
&\le \frac{1}{n}\sum_{i\in[n]}\|\nabla F_i(x_{t-1}) - \nabla F_i(\tilde{x}_{i,t-1})\|^2 \\
&\le L^2\frac{1}{n}\sum_{i\in[n]}\|x_{t-1} - \tilde{x}_{i,t-1}\|^2 && \triangleright\ L\text{-smoothness} \\
&= L^2\frac{1}{n}\sum_{i\in[n]}\left\|\eta\frac{1}{n}\sum_{j\in[n]}(g_{j,t-1}) - \eta\left(DC(g_{t-2} - \frac{1}{n}g_{i,t-2}, \Delta x_{t-2}) + \frac{1}{n}g_{i,t-1}\right)\right\|^2 \\
&= \eta^2L^2\frac{1}{n}\sum_{i\in[n]}\left\|\frac{1}{n}\sum_{j\neq i}(g_{j,t-1}) - DC(g_{t-2} - \frac{1}{n}g_{i,t-2}, \Delta x_{t-2})\right\|^2 \\
&= \eta^2L^2\frac{1}{n}\sum_{i\in[n]}\left\|\frac{1}{n}\sum_{j\neq i}(g_{j,t-1}) - DC\left(\frac{1}{n}\sum_{j\neq i}(g_{j,t-2}), \Delta x_{t-2}\right)\right\|^2 \\
&= \eta^2L^2\frac{1}{n}\sum_{i\in[n]}\left\|\frac{1}{n}\sum_{j\neq i}(g_{j,t-1}) - \left[\frac{1}{n}\sum_{j\neq i}(g_{j,t-2}) + \lambda\left(\frac{1}{n}\sum_{j\neq i}(g_{j,t-2})\right)\left(\frac{1}{n}\sum_{j\neq i}(g_{j,t-2})\right)^\top(x_{t-2} - x_{t-3})\right]\right\|^2 \\
&= \underbrace{2\eta^2L^2\frac{1}{n}\sum_{i\in[n]}\left\|\frac{1}{n}\sum_{j\neq i}(g_{j,t-1}) - \left[\frac{1}{n}\sum_{j\neq i}(g_{j,t-2} + \nabla^2 f(x_{t-3}, z_{j,t-3})(x_{t-2} - x_{t-3}))\right]\right\|^2}_{①} \\
&\quad + \underbrace{2\eta^2L^2\frac{1}{n}\sum_{i\in[n]}\left\|\frac{1}{n}\sum_{j\neq i}\nabla^2 f(x_{t-3}, z_{j,t-3})(x_{t-2} - x_{t-3}) - \lambda\left(\frac{1}{n}\sum_{j\neq i}(g_{j,t-2})\right)\left(\frac{1}{n}\sum_{j\neq i}(g_{j,t-2})\right)^\top(x_{t-2} - x_{t-3})\right\|^2}_{②}.
\end{aligned}
$$

For ①, we have

$$
\begin{aligned}
&① \\
&= 2\eta^2L^2\frac{1}{n}\sum_{i\in[n]}\left\|\frac{1}{n}\sum_{j\neq i}(g_{j,t-1}) - \left[\frac{1}{n}\sum_{j\neq i}(g_{j,t-2} + \nabla^2 f(x_{t-3}, z_{j,t-3})(x_{t-2} - x_{t-3}))\right]\right\|^2 \\
&= 2\eta^2L^2\frac{1}{n}\sum_{i\in[n]}\left\|\frac{1}{n}\sum_{j\neq i}\nabla f(x_{t-2}, z_{j,t-2}) - \left[\frac{1}{n}\sum_{j\neq i}\nabla f(x_{t-3}, z_{j,t-3}) + \nabla^2 f(x_{t-3}, z_{j,t-3})(x_{t-2} - x_{t-3})\right]\right\|^2 \\
&\le 4\eta^2L^2\frac{1}{n}\sum_{i\in[n]}\left\|\frac{1}{n}\sum_{j\neq i}\nabla f(x_{t-2}, z_{j,t-2}) - \frac{1}{n}\sum_{j\neq i}\nabla f(x_{t-2}, z_{j,t-3})\right\|^2 \\
&\quad + 4\eta^2L^2\frac{1}{n}\sum_{i\in[n]}\left\|\frac{1}{n}\sum_{j\neq i}\nabla f(x_{t-2}, z_{j,t-3}) - \left[\frac{1}{n}\sum_{j\neq i}\nabla f(x_{t-3}, z_{j,t-3}) + \nabla^2 f(x_{t-3}, z_{j,t-3})(x_{t-2} - x_{t-3})\right]\right\|^2
\end{aligned}
$$

$$
\begin{aligned}
&\leq 4\eta^2 L^2 \frac{1}{n} \sum_{i\in[n]} \left\| \frac{1}{n}\sum_{j\neq i} \nabla f(x_{t-2}, z_{j,t-2}) - \frac{1}{n}\sum_{j\neq i} \nabla f(x_{t-2}, z_{j,t-3}) \right\|^2 + 4\eta^2 L^2 M^2 \|x_{t-2} - x_{t-3}\|^4 \\
&\leq 4\eta^2 L^2 \frac{1}{n} \sum_{i\in[n]} \left\| \frac{1}{n}\sum_{j\neq i} \nabla f(x_{t-2}, z_{j,t-2}) - \mathbb{E}[\frac{1}{n}\sum_{j\neq i} \nabla f(x_{t-2}, z_{j,t-2})] \right\|^2 \\
&\quad + 4\eta^2 L^2 \frac{1}{n} \sum_{i\in[n]} \left\| \mathbb{E}[\frac{1}{n}\sum_{j\neq i} \nabla f(x_{t-2}, z_{j,t-3})] - \frac{1}{n}\sum_{j\neq i} \nabla f(x_{t-2}, z_{j,t-3}) \right\|^2 \\
&\quad + 4\eta^2 L^2 M^2 \|x_{t-2} - x_{t-3}\|^4 \\
&\leq 8\eta^2 L^2 \frac{V_1}{n} + 4\eta^2 L^2 M^2 \|x_{t-2} - x_{t-3}\|^4 \\
&\leq 8\eta^2 L^2 \frac{V_1}{n} + 4\eta^2 L^2 M^2 \|\eta g_{t-2}\|^4 \\
&\leq 8\eta^2 L^2 \frac{V_1}{n} + 4\eta^6 L^2 M^2 V_2^2.
\end{aligned}
$$

For ②, we have

②

$$
\begin{aligned}
&= 2\eta^2 L^2 \frac{1}{n} \sum_{i\in[n]} \left\| \frac{1}{n}\sum_{j\neq i} \nabla^2 f(x_{t-3}, z_{j,t-3})(x_{t-2} - x_{t-3}) - \lambda \left(\frac{1}{n}\sum_{j\neq i}(g_{j,t-2})\right)\left(\frac{1}{n}\sum_{j\neq i}(g_{j,t-2})\right)^\top (x_{t-2} - x_{t-3}) \right\|^2 \\
&\leq 2\eta^2 L^2 (1-\lambda)^2 \frac{1}{n} \sum_{i\in[n]} \left\| \frac{1}{n}\sum_{j\neq i} \nabla^2 f(x_{t-3}, z_{j,t-3})(x_{t-2} - x_{t-3}) \right\|^2 \\
&\quad + 2\eta^2 L^2 \lambda^2 \frac{1}{n} \sum_{i\in[n]} \left\| \frac{1}{n}\sum_{j\neq i} \nabla^2 f(x_{t-3}, z_{j,t-3})(x_{t-2} - x_{t-3}) - \left(\frac{1}{n}\sum_{j\neq i}(g_{j,t-2})\right)\left(\frac{1}{n}\sum_{j\neq i}(g_{j,t-2})\right)^\top (x_{t-2} - x_{t-3}) \right\|^2 \\
&\leq 2\eta^2 L^2 (1-\lambda)^2 L^2 \left\|x_{t-2} - x_{t-3}\right\|^2 + 2\eta^2 L^2 \lambda^2 \Delta^2 \left\|x_{t-2} - x_{t-3}\right\|^2 \\
&\leq (2\eta^2 L^4 (1-\lambda)^2 + 2\eta^2 L^2 \lambda^2 \Delta^2) \|\eta g_{t-2}\|^2 \\
&\leq 2\eta^4 L^2 V_2 (L^2 (1-\lambda)^2 + \lambda^2 \Delta^2).
\end{aligned}
$$

Putting all the ingredients together, we have

$$
\begin{aligned}
&\mathbb{E}[F(x_t)] \\
&\leq F(x_{t-1}) - \frac{\eta}{2}\|\nabla F(x_{t-1})\|^2 \\
&\quad + \frac{\eta}{2}\left\|\nabla F(x_{t-1}) - \mathbb{E}[g_t]\right\|^2 + \frac{L\eta^2}{2}\mathbb{E}\left[\|g_t - \mathbb{E}[g_t]\|^2\right] \\
&\leq F(x_{t-1}) - \frac{\eta}{2}\|\nabla F(x_{t-1})\|^2 \\
&\quad + \frac{\kappa\eta}{2}[8\eta^2 L^2 \frac{V_1}{n} + 4\eta^6 L^2 M^2 V_2^2 + 2\eta^4 L^2 V_2 (L^2(1-\lambda)^2 + \lambda^2\Delta^2)] + \frac{L\eta^2}{2}\frac{V_1}{n}.
\end{aligned}
$$

By re-arranging the terms, we have

$$
\begin{aligned}
&\|\nabla F(x_{t-1})\|^2 \\
&\leq \frac{2\mathbb{E}[F(x_{t-1}) - F(x_t)]}{\eta} \\
&\quad + \kappa(8\eta^2 L^2 \frac{V_1}{n} + 4\eta^6 L^2 M^2 V_2^2 + 2\eta^4 L^2 V_2 (L^2(1-\lambda)^2 + \lambda^2\Delta^2)) + \frac{L\eta V_1}{n}.
\end{aligned}
$$

By telescoping and taking total expectation, after T iterations, we have

$$
\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla F(x_{t-1})\|^2\right] \\
&\leq \frac{2\mathbb{E}[F(x_0) - F(x_*)]}{\eta T} \\
&\quad + \kappa(8\eta^2 L^2 \frac{V_1}{n} + 4\eta^6 L^2 M^2 V_2^2 + 2\eta^4 L^2 V_2 (L^2(1-\lambda)^2 + \lambda^2\Delta^2)) + \frac{L\eta V_1}{n}.
\end{aligned}
$$

□

B Proof of Theorem 1

Theorem 1. *In training pipeline with no stall, if some forward layer is stale, then all its preceding forward layers are stale.*

Proof. We prove this by contradiction. Given staleness for i-th forward layer, it satisfies the layer stale condition:

$$c(u_i) < c(v_i) + v_i. \tag{2}$$

Suppose there exists k-th layer which is not stale, where $k < i$. According to comm-forward dependency condition, we have

$$c(u_k) \geq c(v_k) + v_k. \tag{3}$$

Since $k < i$, according to forward dependency, we have

$$c(u_k) < c(u_i). \tag{4}$$

Under no preemption assumption, $c(v_i) + v_i \leq c(v_k)$. Combine equations 4 3, we have

$$c(u_i) > c(u_k) \geq c(v_k) + v_k \geq c(v_i) + v_i + v_k > c(v_i) + v_i,$$

which is in contradiction with equation 2. □

C Optimized Runtime

C.1 Weight Update

Overhead of many small kernel launch. The default implementation of the weight update function in the ML framework backend involves multiple CUDA kernel launch overhead for each independent parameter. The weight update occurs upon the arrival of each synchronized gradient, which leads to many fragments of small kernels for each parameter. Each kernel launch, regardless of the real computation complexity, incurs some fixed overhead, such as kernel latency, kernel overhead, CPU launch overhead and additional overhead [32]. In general, these kernels cannot be fused together considering different completion time of gradients, while setting a barrier for the completion of all gradient synchronization incurs large waiting overhead. This issue is enlarged with staleness mitigation methods, which introduce extra weight updates and multiple staleness data movement.

Our Solution: Kernel Fusion. However, for the staleness pipeline, the weight update kernels could be fused since the input tensors are stored in the staleness buffer in the last step. After backward propagation, the communication operations are executed to synchronize the gradients. The results of synchronized stale gradients are stored in the staleness buffer, while others are directly applied to update the weights on arrival. And the results of stale gradients in the last step are retrieved and the corresponding weight updates kernels are fused together to reduce the overhead.

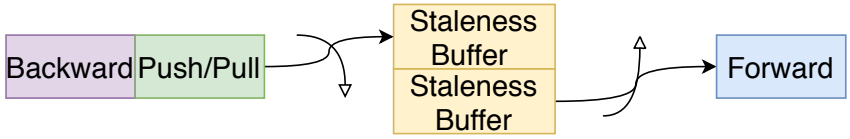
C.2 Double Buffer Optimization

Extra buffer overhead. There are two staleness buffers for each parameter in SAPipe to enable the cross-iteration execution: the read buffer for computation pipeline reading gradients from previous step and the write buffer for communication pipeline writing synchronized gradients in current step. The double staleness buffer setup increases the GPU memory consumption and incur memory copy overhead to update the reading buffer in each iteration.

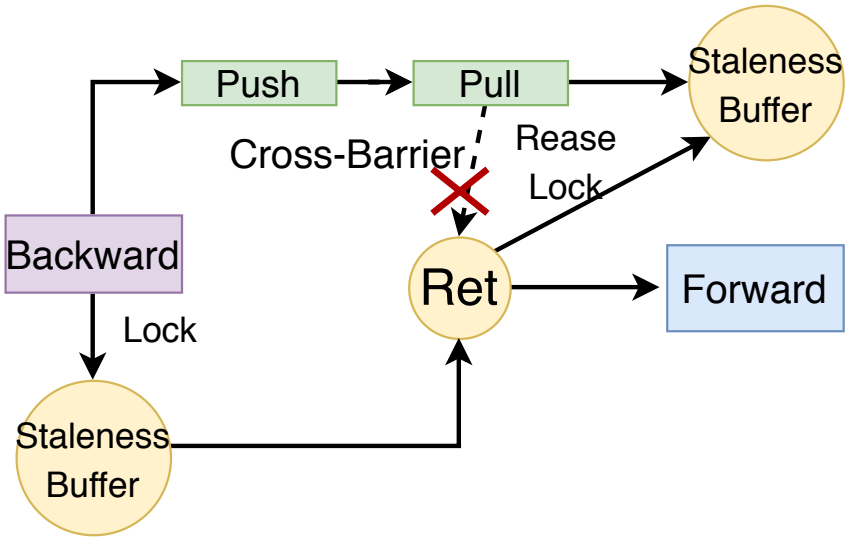
Our Solution: Double Buffer System and Buffer Sharing. We adopt *double buffer system* technique to omit the memory copy overhead between double staleness buffers. The reading and writing of the staleness buffers shuffles at each training step. The communication service alternate different buffers to write the synchronization results, while the forward computation can read stored data from the available buffer at each step without colliding. This omits the memory copy overhead between two buffers used in different pipeline stages.

To further minimize the memory consumption of staleness buffer, we explore to share the read and write buffer between communication and computation stages. When the read buffer finishes passing the previous step's gradient to the computation backend [6], the synchronized gradient in current step

[6] The buffer reading time here denotes the time interval between backward and forward propagation, which consists of dependency awaiting and data transferring time.



(a) Double buffer system



(b) Buffer sharing

Figure 6: Optimize staleness buffer.

could be allowed to update the reading buffer without being stored in extra memory (see Figure 6 (b)). A dependency is added to avoid race condition that the buffer could be updated only after it has been read in current step. However, this dependency may cause waiting time in the communication pipeline, harming the throughput potentially. Hence, we share the staleness buffer only for the communication-intensive tasks, where the communication time is larger than the buffer reading time.

C.3 Buffer Switch

Transferring overhead of staleness buffer. The staleness buffers are involved in both communication and computation pipelines: 1) the results of synchronized gradient are written into the staleness buffer; 2) the computation stage reads gradients from the staleness buffer for parameter updates. Since the NIC is connected to the PCIe of one of the CPU socket and the computation is on the GPU device, there exists transferring overhead between host and device memory even with only one staleness buffer. The buffer transferring overhead between CPU and GPU is not negligible, considering the slow PCIe bandwidth and large number of parameters.

Our Solution: Buffer Switch. However, the buffer transferring overhead between CPU and GPU can be hidden behind pipeline stages of SAPipe. The communication and computation pipelines pipeline of SAPipe might not be balanced due to different communication-to-computation ratios for DNN training jobs. Hence, the staleness buffer could be placed close to the faster pipeline stage for balanced training pipeline. To reduce the transferring overhead, we dynamically switch the location of the staleness buffer according to the duration of communication and computation pipeline stages. Given a DNN training job, SAPipe will profile the completion time of communication (v_i)and computation (b_i and u_i) through warm-up steps, as well as the buffer transferring overhead δ, and determine the location of the staleness buffer for all gradients: GPU buffer for low communication-to-computation ratio (*e.g.*, $\frac{\sum_i v_i+\delta}{\sum_i(b_i+u_i)} \leq 1$), and CPU buffer for the opposite case (*e.g.*, $\frac{\sum_i v_i}{\sum_i(b_i+u_i)+\delta} > 1$).

The intuition behind is to remove the transferring overhead from slower pipeline stage so as to balance the training pipeline.

Algorithm 4 Determine the start time of comm ops.

Input: Forward DAG dependency graph $G = (B, V, E)$, where b_n is the first backward layer;
Output: Start time $c(v_i)$ of communication operators.

1: $S \leftarrow \{b_m\}$
2: $t \leftarrow c(b_m)$
3: $r(c_m) \leftarrow 0$
4: **while** S is not empty **do**
5: $\quad b_i \leftarrow \text{RANDOMSELECT}(S)$
6: $\quad S \leftarrow S - \{b_i\}, S \leftarrow S + \{b_j | (b_i, b_j) \in E\}$
7: $\quad r(v_i) \leftarrow t$
8: $\quad t \leftarrow t + b_i$
9: $R \leftarrow \text{SORT}(r(c_1), \ldots, r(v_m))$ // in descending order
10: $t \leftarrow 0$
11: **while** R is not empty **do**
12: $\quad r(v_k) \leftarrow R[-1]$
13: $\quad S \leftarrow R.where(r(v_i) <= t) + v_k$
14: $\quad i = \text{GETMININDEX}(S)$
15: $\quad S \leftarrow S - v_i$
16: $\quad c(v_i) \leftarrow t$
17: $\quad t \leftarrow t + v_i$
18: $\quad R \leftarrow R - r(v_i)$

D Partial Staleness on DAG Model

In a more general case where the forward and backward propagation cannot be linearized as a chain of computation across layers, Theorem 1 does not hold and the searching algorithm for sequential models falls short in finding optimal solution. Solving the optimal stale gradient problem for general DAG models is NP-hard. Hence, we present the prototypical heuristic for finding the partial stale gradients for DAG model, as shown in Algorithm 5.

Algorithm 4 determines the execution order and start time for each communication operation, and Algorithm 5 finds the optimal solution of stale gradients. Line 1-8 of Algorithm 4 output the time of each communication operation that is ready to be launched, which is the completion time of their corresponding backward operations. By default, the execute order of available backward operations is decided randomly as in Line 5. Line 9 sorts the ready time of communication operations in descending order, and Line 5-18 determines the launching time for all communication operations. Line 13 selects the candidate communication operations that are available at current scheduling time unit, and Line 14 determines the one with minimal index in ready time set to be launched.

Algorithm 5 further determines the staleness of all gradients according to the scheduled communication operations and forward computation. Line 4 randomly selects one available forward operation u_i from set S, and Line 5 updates set S with the neighbors of u_i. Line 6-10 checks the stale condition of selected forward operation u_i: if its dependent communication operation v_i has not been finished at the scheduling time, then v_i is stale; otherwise, if v_i is finished, u_i can be computed immediately and v_i is non-stale.

E Discussion

SAPipe has some requirements for superior model performance:

• Our algorithm for selecting partial staleness can only be used in sequential models, though the staleness compensation and runtime optimizations can be directly applied to a more complicated DAG model. Extending SAPipe to complicated DAG models is our future work.

• The throughput improvement depends on the communication-to-computation ratios. When training models with relatively higher (but no greater than 1) communication-to-computation ratios, SAPipe can achieve higher speedups than the baseline due to higher overlapping potentials.

Algorithm 5 Searching partial staleness for DAG.

Input: Forward DAG dependency graph $G = (U, E)$, where u_1 is the first forward layer; starting time for communication operators $c(v_i), \forall i = 1, \ldots, m$; starting time for the first forward layer $c(u_1)$

Output: Partial staleness for each layer, $x_i, \forall i = 1, \ldots, m$.

$S \leftarrow \{u_1\}$
$t \leftarrow c(u_1)$
while S is not empty **do**
    $u_i \leftarrow \text{RANDOMSELECT}(S)$
    $S \leftarrow S - \{u_i\}$, $S \leftarrow S + \{u_j | (i, j) \in E\}$
    $t \leftarrow t + u_i$
    if $t < c(v_i)$ **then**
        $x_i \leftarrow 1$
    else
        $x_i \leftarrow 0$

• Theoretically, many factors can affect accuracy, which depends on the properties of the datasets and model structures. Our methods can achieve higher performance under certain conditions, e.g., low gradient variance, low gradient diversity, and good smoothness (as discussed in Remark 7 of Section 4.2).

• The performance of staleness mitigation methods varies in different models and datasets. The best choice of the mitigation methods depends on the choice of hyperparameters and some unknown constant values such as smoothness, gradient diversity and variance. This is the limitation and future work of our paper.

F Experiments

F.1 Datasets Details

We train CV models on two datasets: (1) CIFAR-10 dataset [16], which consists of 50,000 training images and 10,000 test images with 10 classes, and (2) ImageNet dataset [17], which contains 1,281,167 training and 50,000 validation images with 1000 classes. We fine-tunes the pretrained GPT-2 model on (3) WikiText-2 language modeling dataset [20], which is a collection of over 2 million tokens extracted from the set of articles on WikiPedia. Transformer model is trained on (4) Multi30K dataset [8], which is a English-to-German multimodel translation dataset with 29,000 training and 1,000 test sentences.

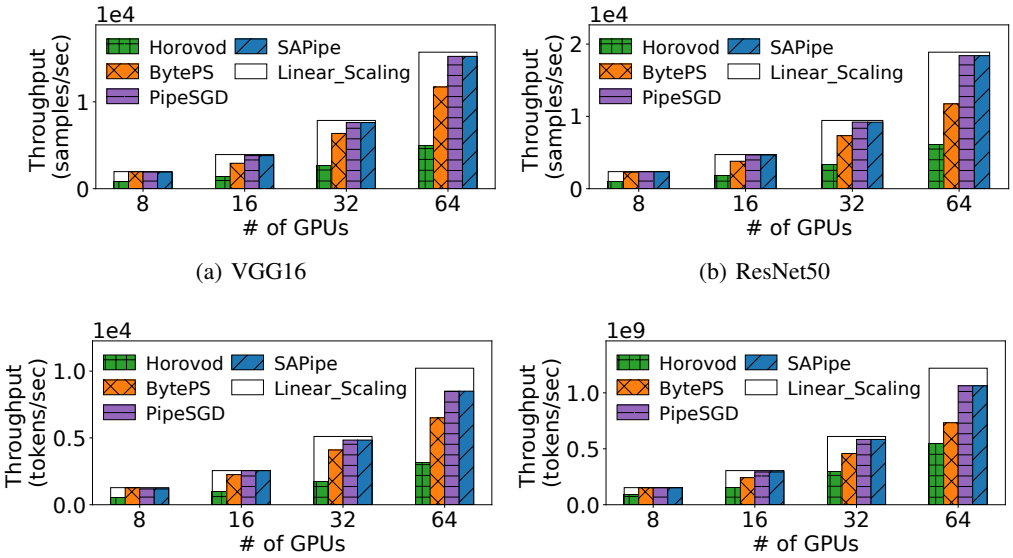
F.2 Magnified Figures and Extra Experiments

As shown in Figure 9, in most cases, without the proposed staleness mitigation methods, the vanilla PipeSGD has significant regression in accuracy/perplexity compared to the non-stale baseline, when the same number of iterations are executed. Additionally, we see that PipeSGD has much worse converged results than the baseline in NLP models. This may result from the more complicated models, which are more sensitive to the staleness. Overall, SAPipe has comparable converge rate and final converged results across four models.

Figure 10(a) and (b) show the throughput speedup and staleness mitigation with partial staleness. With partial staleness, SAPipe further reduces the negative impact of staleness on model performance, and improves the accuracy by 1.28% and 2.4% for VGG16 and ResNet50, respectively. We observe divergence when training PipeSGD with ResNet50 on CIFAR-10 dataset, while our staleness mitigation methods greatly solves this issue.

F.3 Divergence on PipeSGD

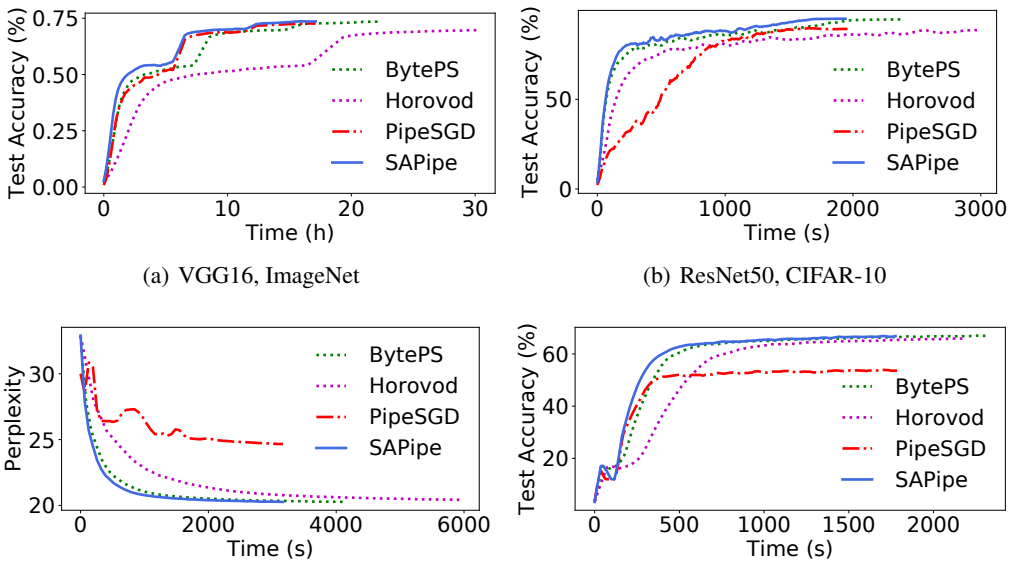
We use 2 GPUs to train VGG16 and ResNet50 on CIFAR-10 dataset, and observe severe convergence problems of PipeSGD, as shown in Figure 11. PipeSGD diverges on ResNet50 model with 1-step staleness, while SAPipe mitigates the staleness problem and converges as fast as the BytePS baseline.



(a) VGG16 (b) ResNet50

(c) GPT-2 (d) Transformer

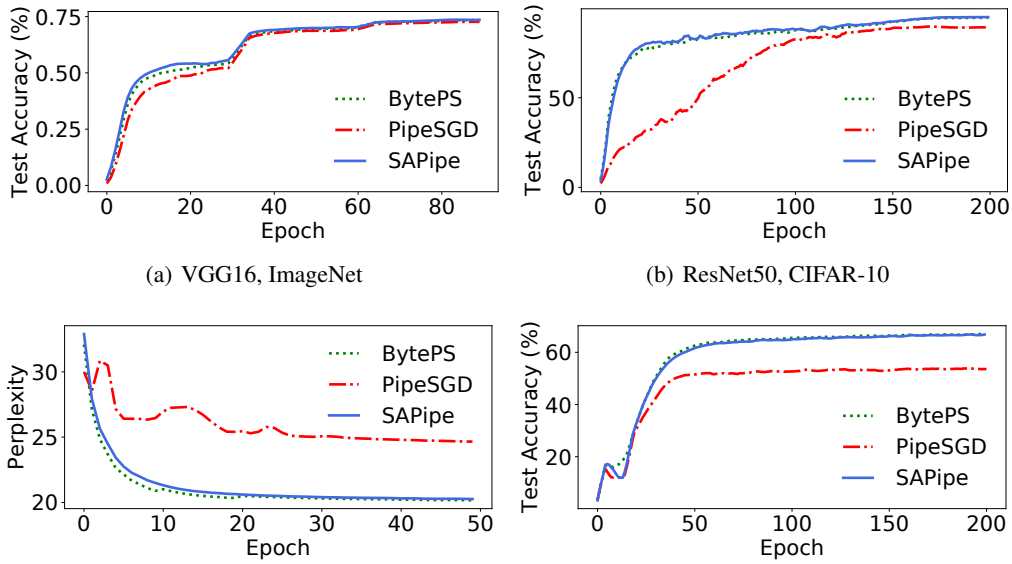
Figure 7: Training throughput.



(a) VGG16, ImageNet (b) ResNet50, CIFAR-10

(c) GPT-2, WikiText-2 (d) Transformer, WMT16

Figure 8: Convergence of different models. The x-axis is wall-clock training time, and the y-axis is perplexity (lower is better) for GPT-2, and test accuracy (higher is better) for others.



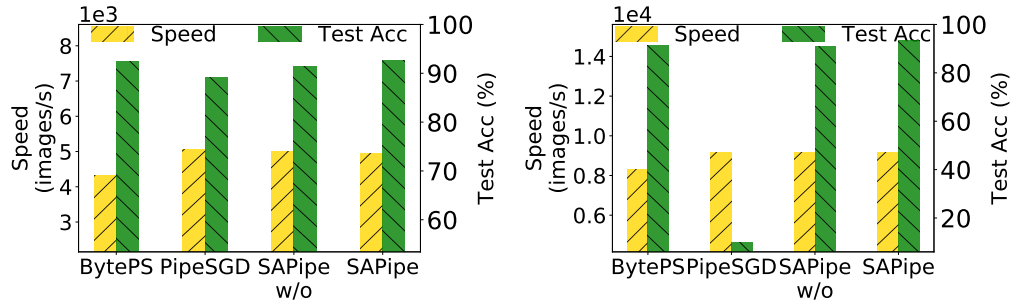
(a) VGG16, ImageNet

(b) ResNet50, CIFAR-10

(c) GPT-2, WikiText-2

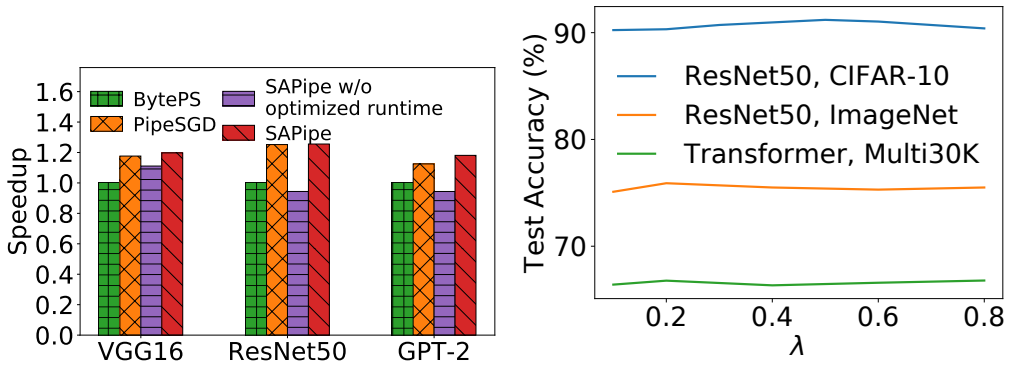
(d) Transformer, WMT16

Figure 9: Convergence of different models. The x-axis is the number of epochs.



(a) VGG16 with partial staleness. "SAPipe w/o" denotes SAPipe without partial staleness.

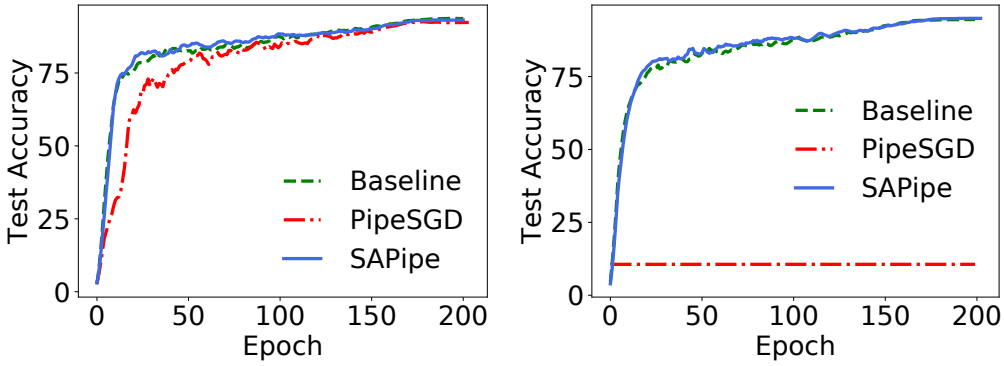
(b) ResNet50 with partial staleness. "SAPipe w/o" denotes SAPipe without partial staleness.



(c) Runtime optimization. "SAPipe w/o" denotes SAPipe without runtime optimization.

(d) Sensitivity analysis

Figure 10: Deep dive in SAPipe.



(a) VGG16

(b) ResNet50

Figure 11: Training on CIFAR-10 with 2 GPUs. X-axis is the training epoch, and y-axis is the test accuracy.