Accelerating Distributed ML Training via Selective Synchronization

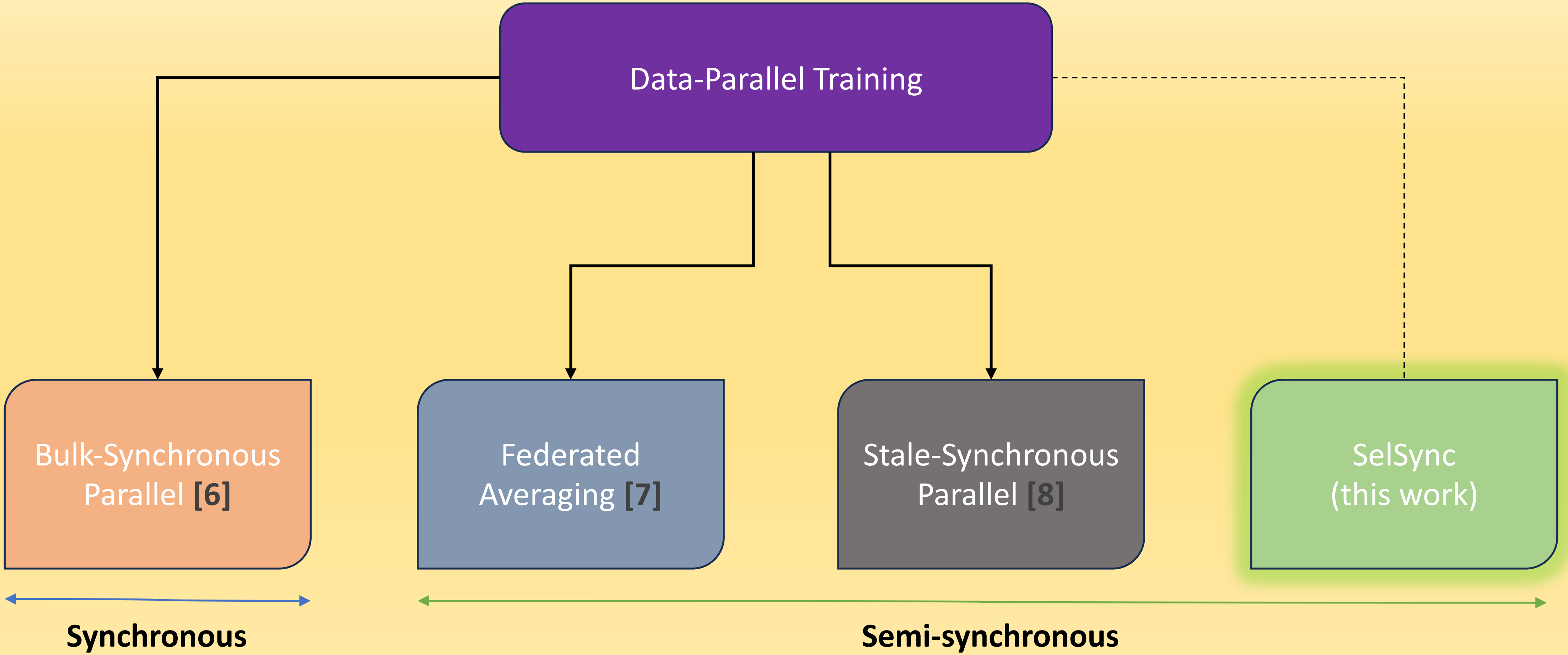
Sahil Tyagi and Martin Swany

Why is Distributed ML Training important?

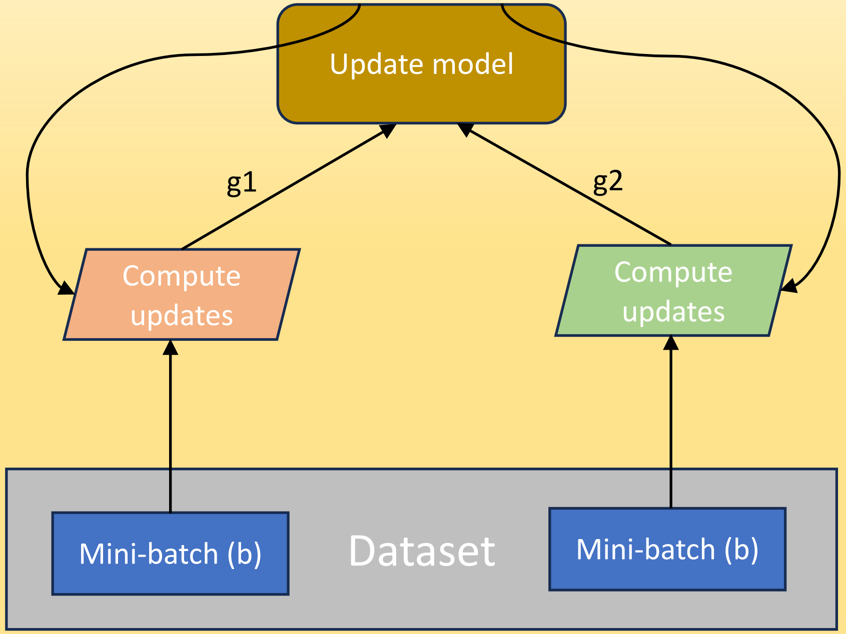
- Exponentially growing size of neural networks in recent years
  - **2020**: BART (140 million), Turing-NLG (17 billion)
  - **2021**: ViT (630 million), DALL-E (12 billion)
  - **2022**: Stable Diffusion (890 million), GPT-3.5 (1.3-175 billion)
  - **2023**: GPT-4 (1.8 trillion)
- Massive repositories of potential training data
- Maintain Data Privacy and security (federated learning)
- Reduce training time and cost/energy of running jobs in the cloud/data-center

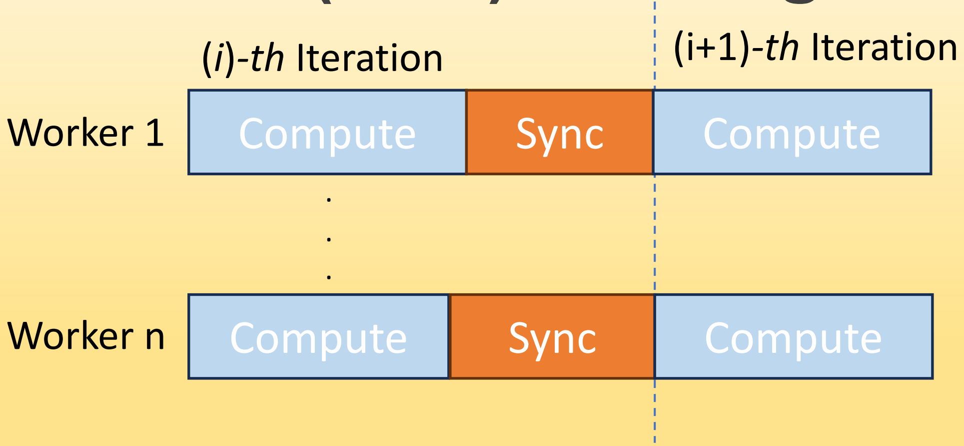
# Params	Params-size (MB)
1e6	4
1e7	40
1e8	400
1e9	4000

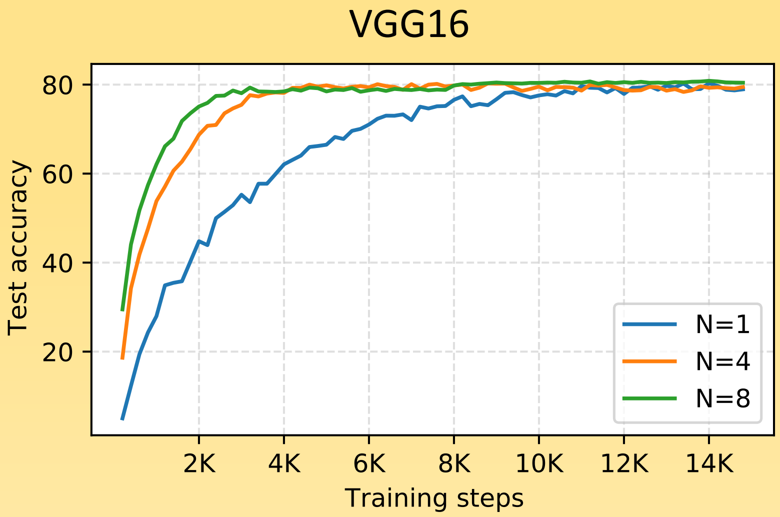
Current Approaches in Distributed Data-Parallel Training



Background: Synchronous Data-Parallel (BSP) Training







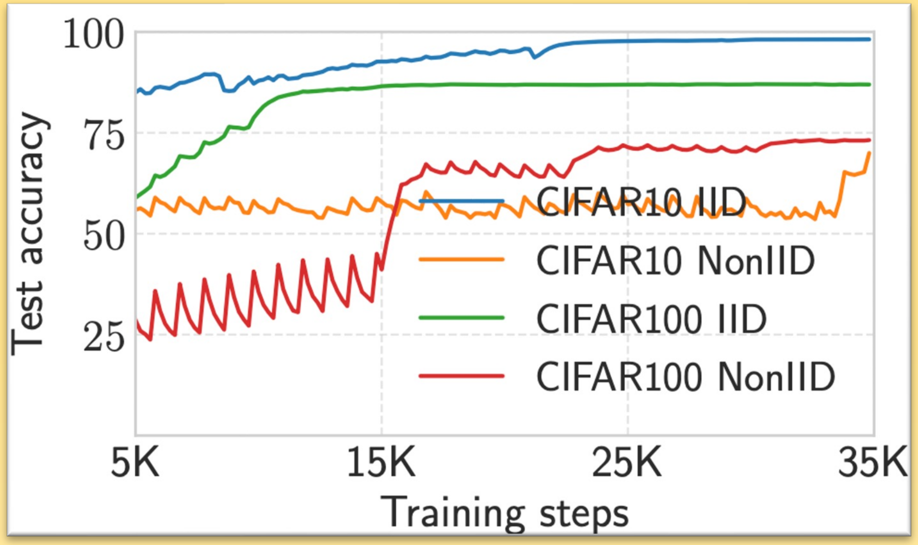
$$w_{i+1} = w_i - \eta \frac{1}{N} \sum_{n=1}^{n=N} \frac{\partial}{\partial w_i} \left(\frac{1}{|b|} \sum_{d_{(i,n)} \in \mathcal{D}_n} \mathcal{L}(x_{(i,n)}, w_i) \right)$$

Background: Federated Averaging

- Federated learning crucial for on-device, data-local training
- **FedAvg [7]** is a low-frequency, high-volume federated learning approach in settings with balanced and unbalanced data distributions
- Updates from fraction of clients (C) aggregated infrequently (E) on a central server. for e.g., **(C, E) = (0.5, 0.25)**

Data distribution significantly affects model convergence!

FedAvg: ResNet101 on CIFAR10 and VGG11 on CIFAR100 with (1, 0.1)

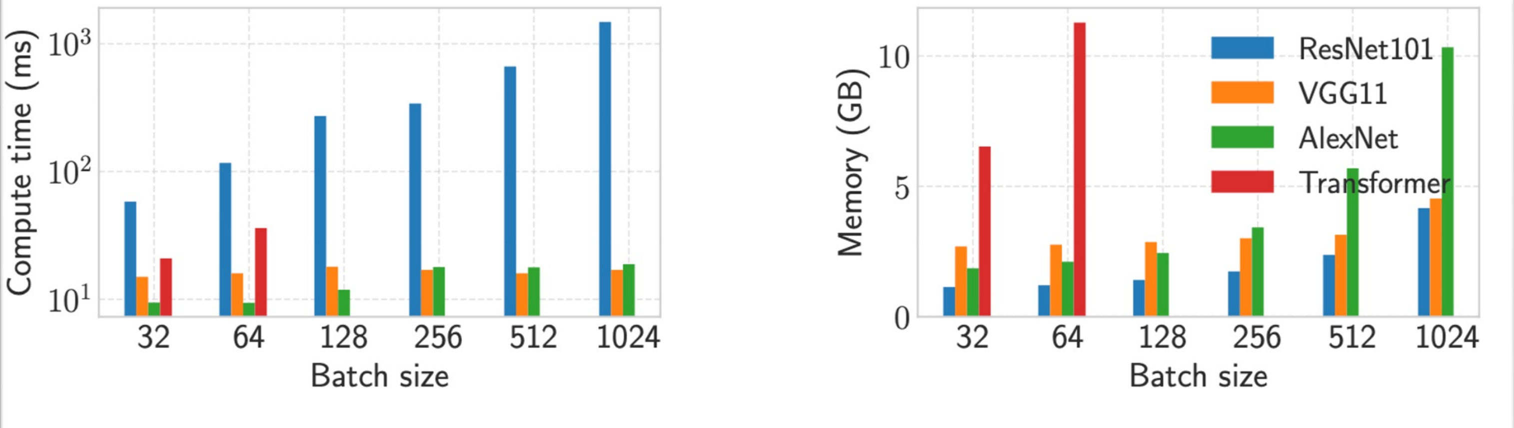


Background: Stale-Synchronous Parallel Training

- **SSP [8]** allows workers to asynchronously send updates to central server
- Asynchronicity is however conditional; determined by ***staleness-threshold*** parameter ‘**s**’
- Parallel scaling can be improved by performing more work per-iteration (using larger batch-sizes)

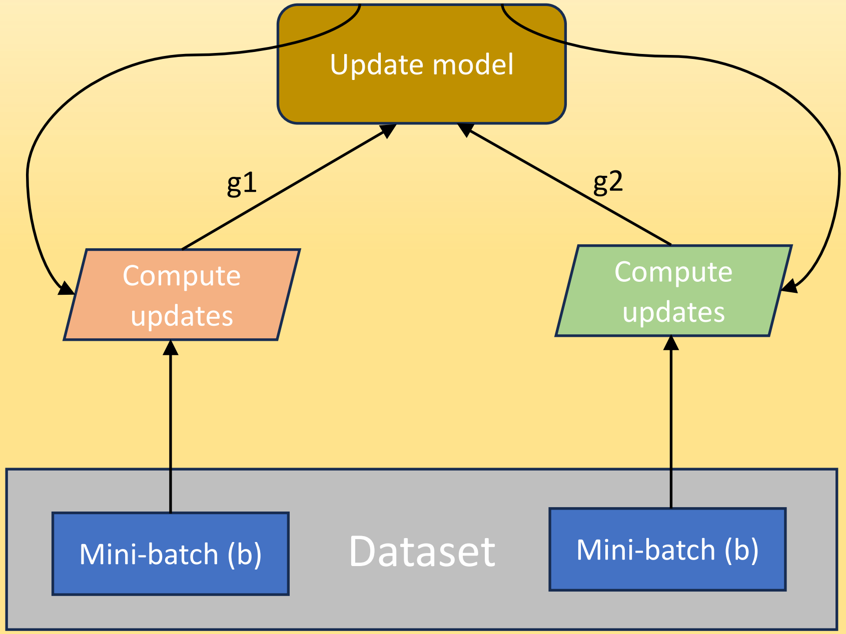
Thus, there are computational limits to how much we can scale SSP proportional to BSP

Improving parallel efficiency of SSP



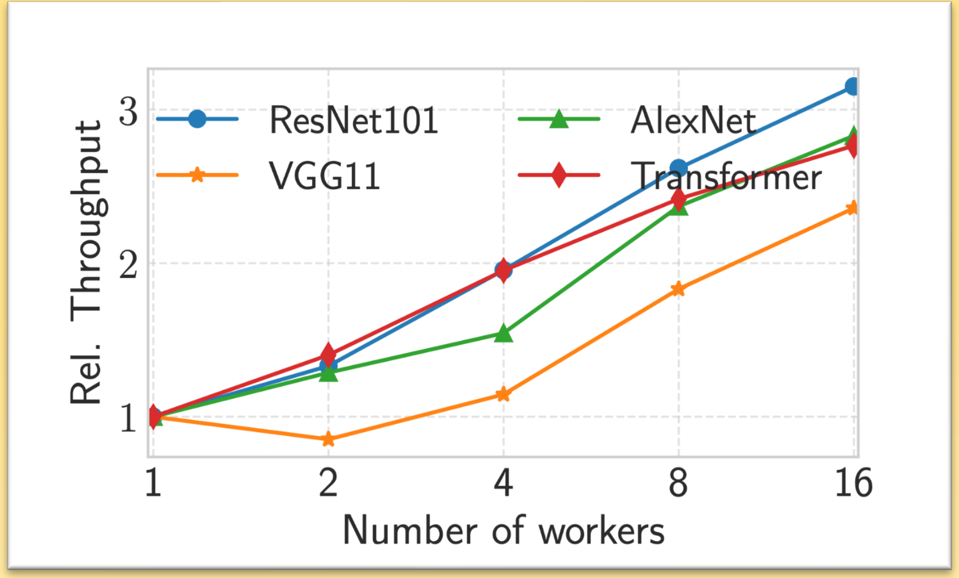
(a) Compute time vs. batch-size (b) Memory util. vs. batch-size

Parallel Efficiency in Distributed Training



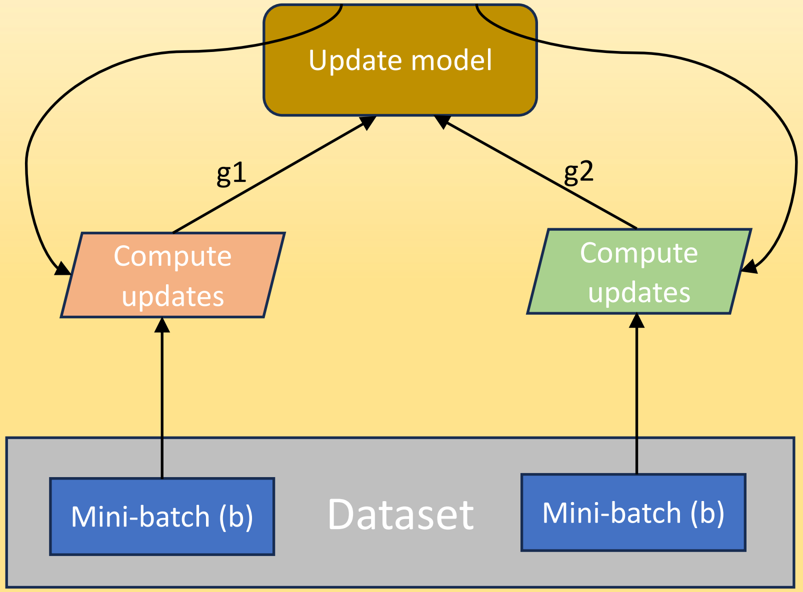
Iteration/Step-time comprised of:

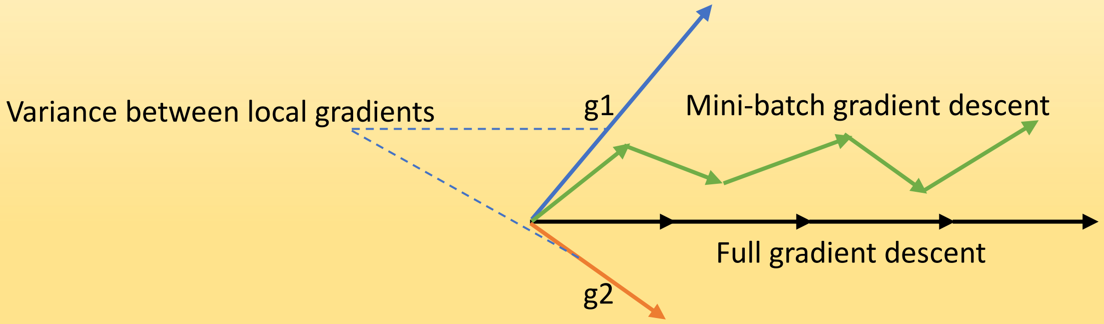
$$t_{step} = t_{compute} + t_{sync} + t_{IO}$$



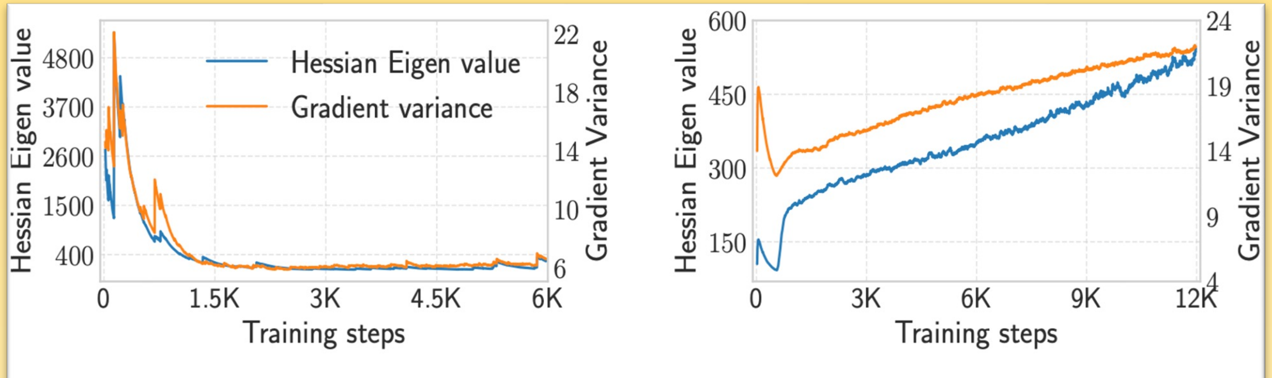
Synchronization cost prevents linear scaling of distributed training jobs and slows convergence [1, 2]

Statistical Efficiency in Distributed Training





First-order information effectively approximates second-order gradients



(a) ResNet101

(b) VGG11

- SGD not fully composable due to its stochastic nature
- Certain training phases or regions are more critical **[3,4,5]**

Summarizing prior methods

- DDP methods either maximize useful work by iterative aggregation of worker updates (BSP) or speedup training by reducing communication frequency (FedAvg) or loosening constraints on synchronization (SSP)

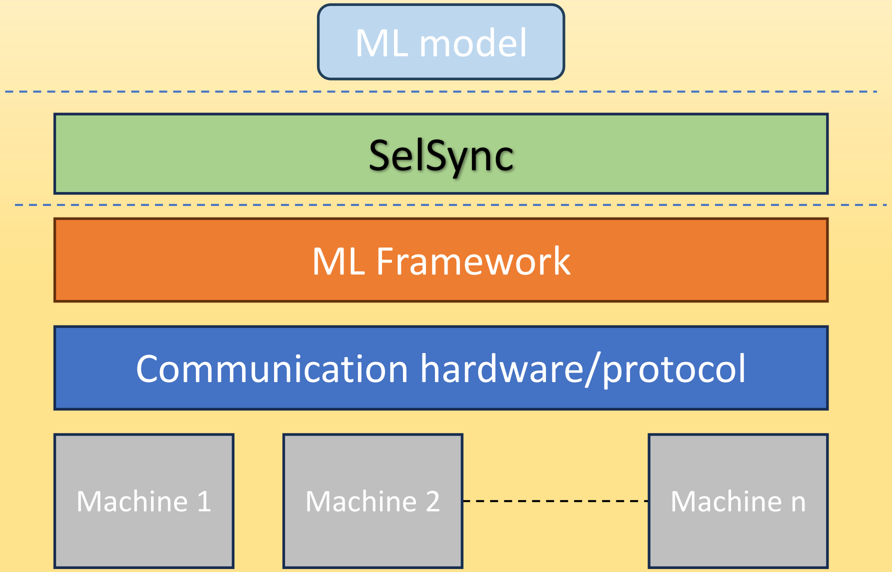
- Compared to BSP, semi-synchronous methods attain significant training speedup

- However, they primarily consider the parallel efficiency and ***not*** the statistical efficiency of distributed training

- **This reflects in the final model accuracy/eval metric of FedAvg and SSP under different (C, E) and staleness-threshold configurations!**

SelSync's approach

- Ideal approach should consider both the parallel and statistical efficiency in distributed training
- Improve *parallel efficiency* by reducing communication cost
- Improve *statistical efficiency* by identifying critical/sensitive sections of training phase followed by synchronization; gradients tend to be more volatile in these regions

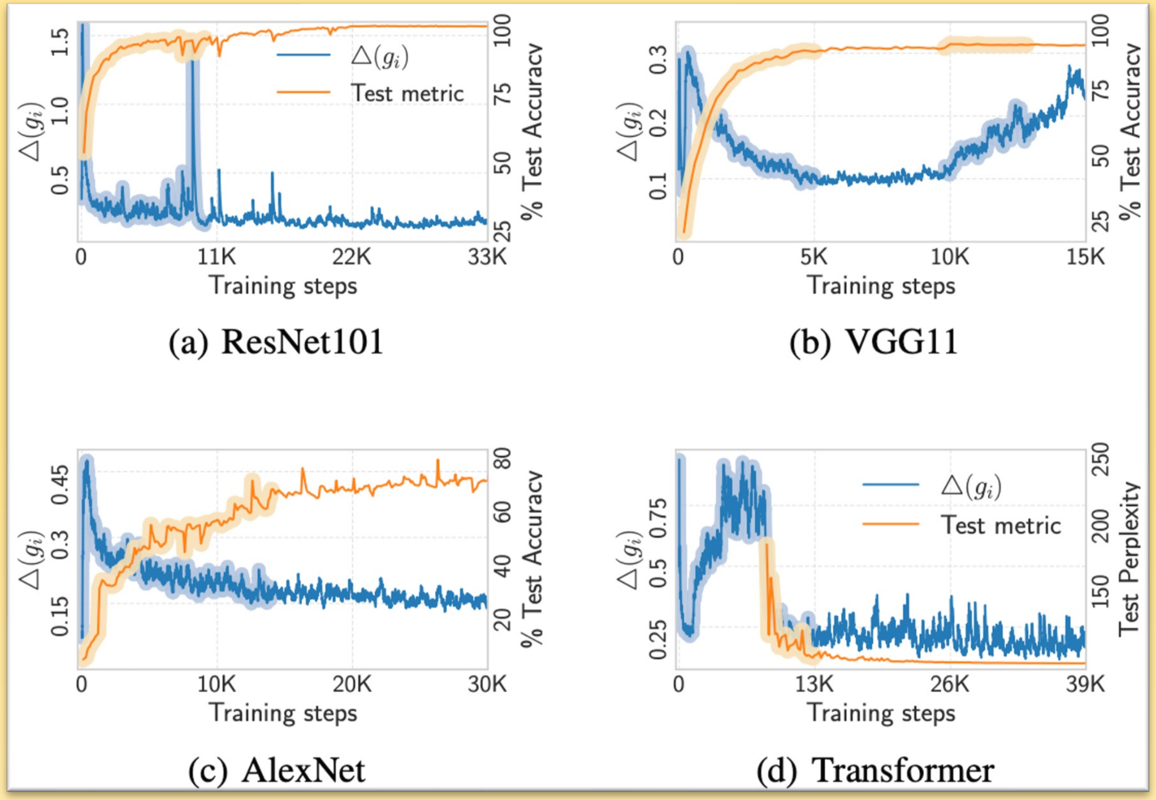


Can we communicate updates among workers only if they are critical/important and avoid expensive synchronization cost when they are not?

SelSync = {Sel}ective {Sync}hronization

SelSync's approach cont'd…

- First-order gradient information works as an effective heuristic to measure significance of model updates; ***measure changes in the variance of inter-iteration gradients***



(a) ResNet101 (b) VGG11

(c) AlexNet (d) Transformer

We define ***Relative Gradient Change*** as:

$$\triangle(g_i) = \left| \frac{\mathbb{E}[||\nabla \mathcal{F}_{(i)}||^2] - \mathbb{E}[||\nabla \mathcal{F}_{(i-1)}||^2]}{\mathbb{E}[||\nabla \mathcal{F}_{(i-1)}||^2]} \right|$$

Delta-based selective synchronization

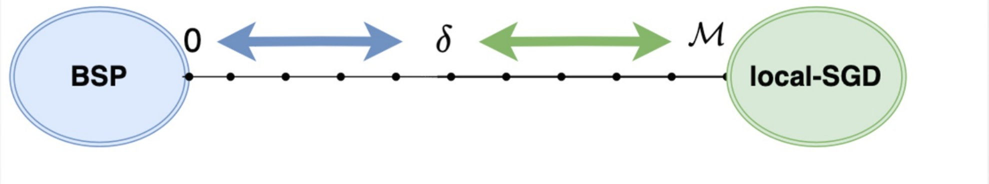


Fig. 6. Adjusting threshold δ on relative gradient change. Choose BSP if $\triangle(g_i) \geq \delta$ and local SGD if $< \delta$. Setting δ=0 implies BSP training, while a very high δ trains only with local updates.

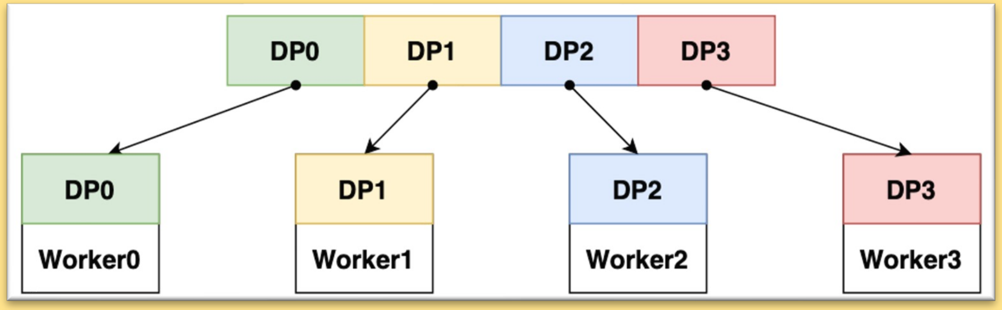
Data-partitioning in Synchronous Training

- In traditional BSP, split dataset ***D*** into ***N*** unique partitions across ***N*** workers

- Referred to as ***Default Data-Partitioning (DefDP)***

- Does not work well in context of semi-synchronous training

DefDP



- **Local models may fail to learn features from data partitions on other workers in settings with low communication and largely local training**

Data-partitioning in Semi-Synchronous Training

- Partitioning scheme optimal for hybrid of local and synchronous updates

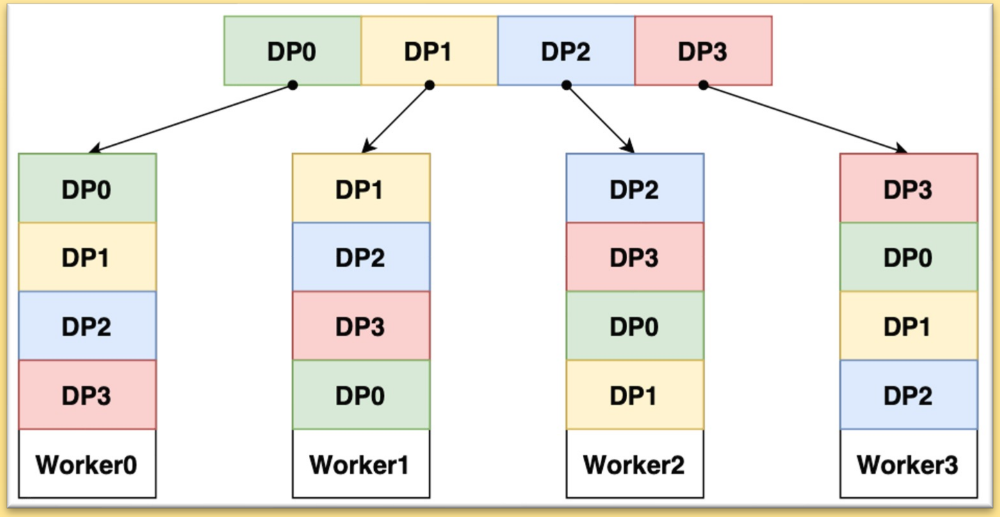
- Instead of partitioning into subset of unique chunks, shuffle chunks of ***D*** based on worker ID

- Referred as **SelSync Data-Partitioning (SelDP)**

- Local model replicas are thus not skewed from ***mostly*** local training

- During synchronization step, each worker update comes from a unique chunk

SelDP



Data-partitioning in Semi-Synchronous Training cont'd…

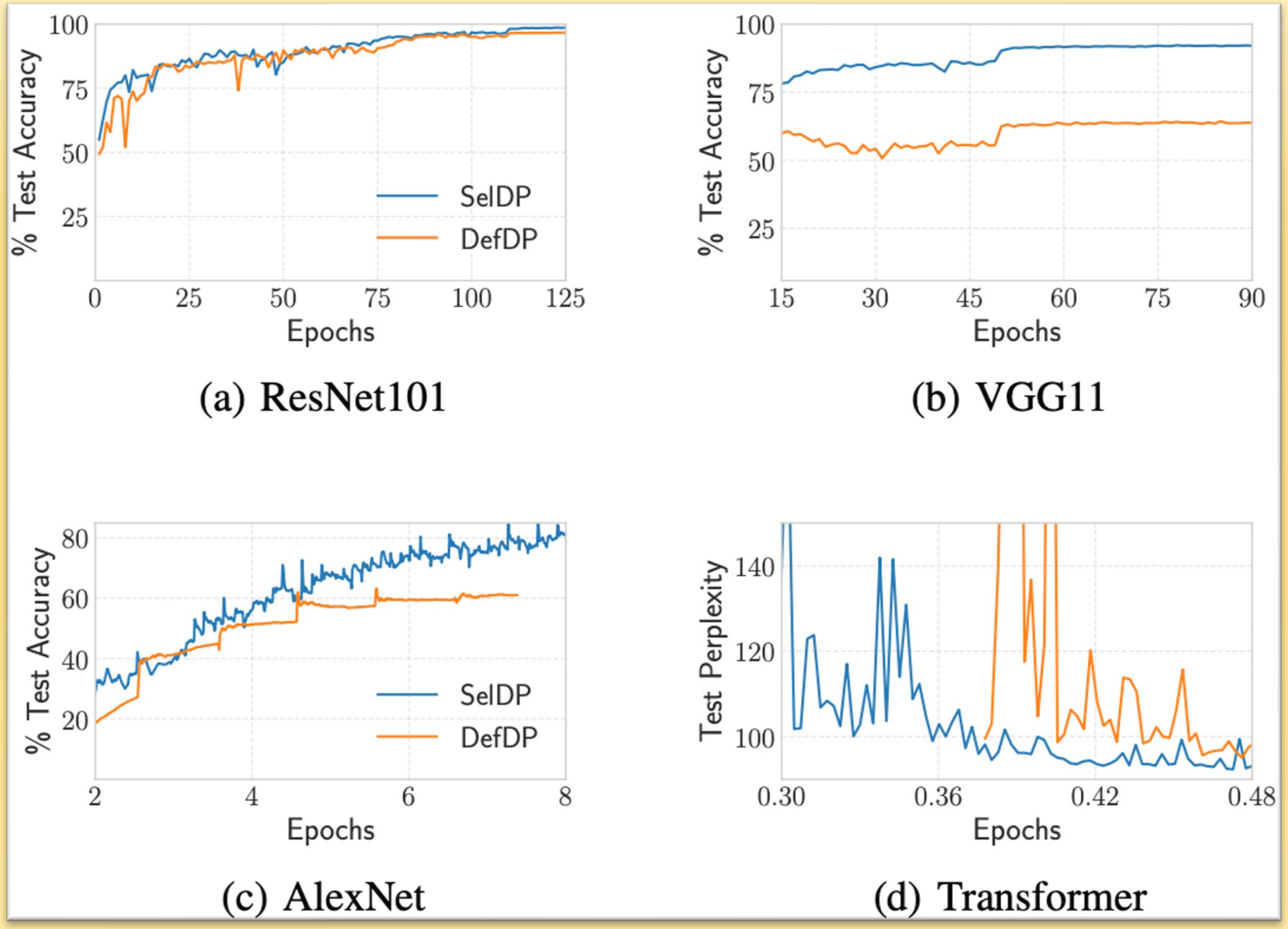
ResNet101 on CIFAR10

VGG11 on CIFAR100

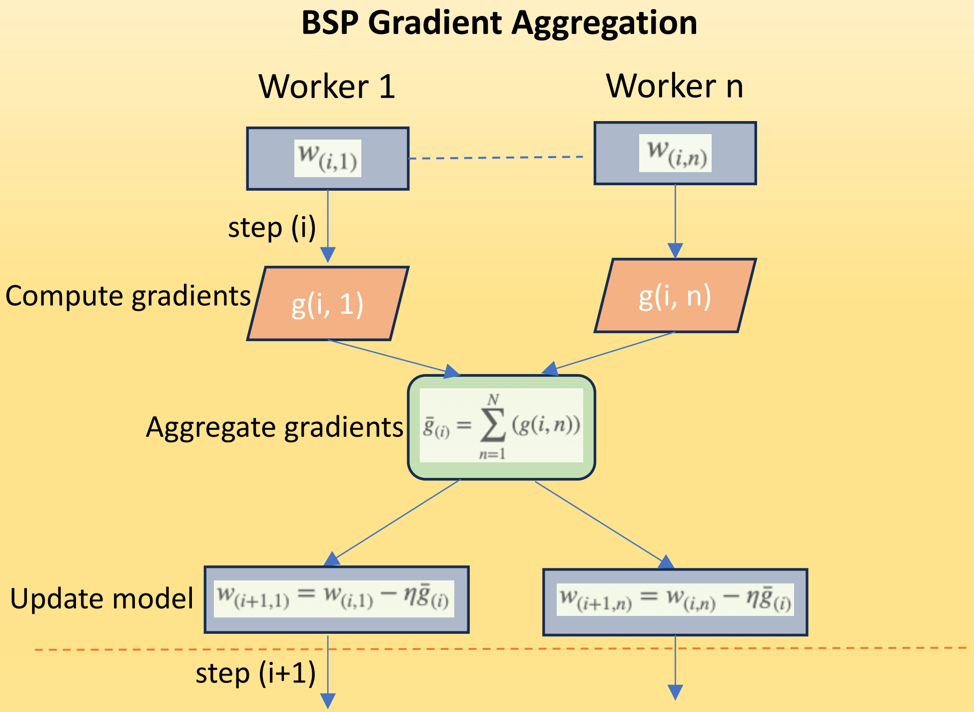
AlexNet on ImageNet-1K

Transformer on WikiText-103

Set delta to 0.25

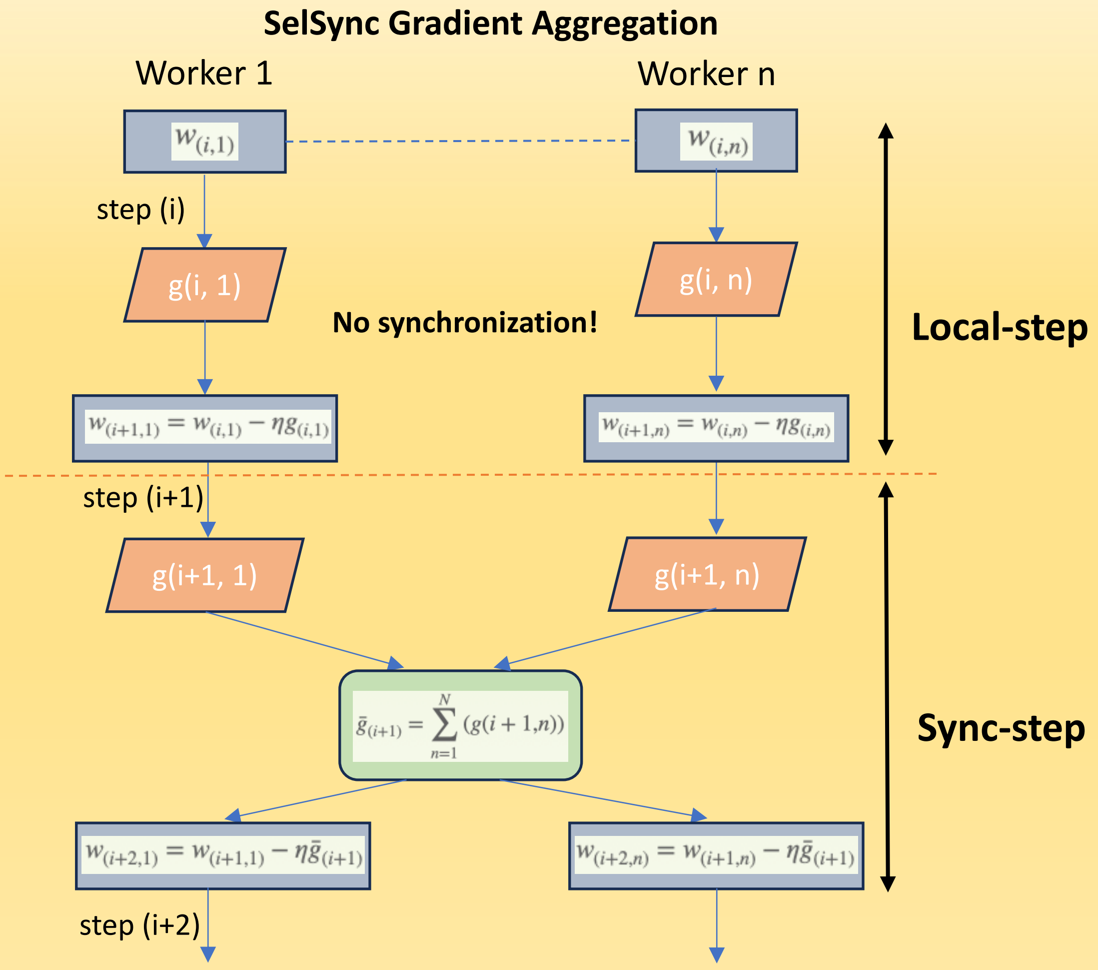


Gradient vs. Parameter Aggregation in SelSync



In BSP, for any iteration *i* $w_{(i,1)} = w_{(i,n)}$ $w_{(i+1,1)} = w_{(i+1,n)}$

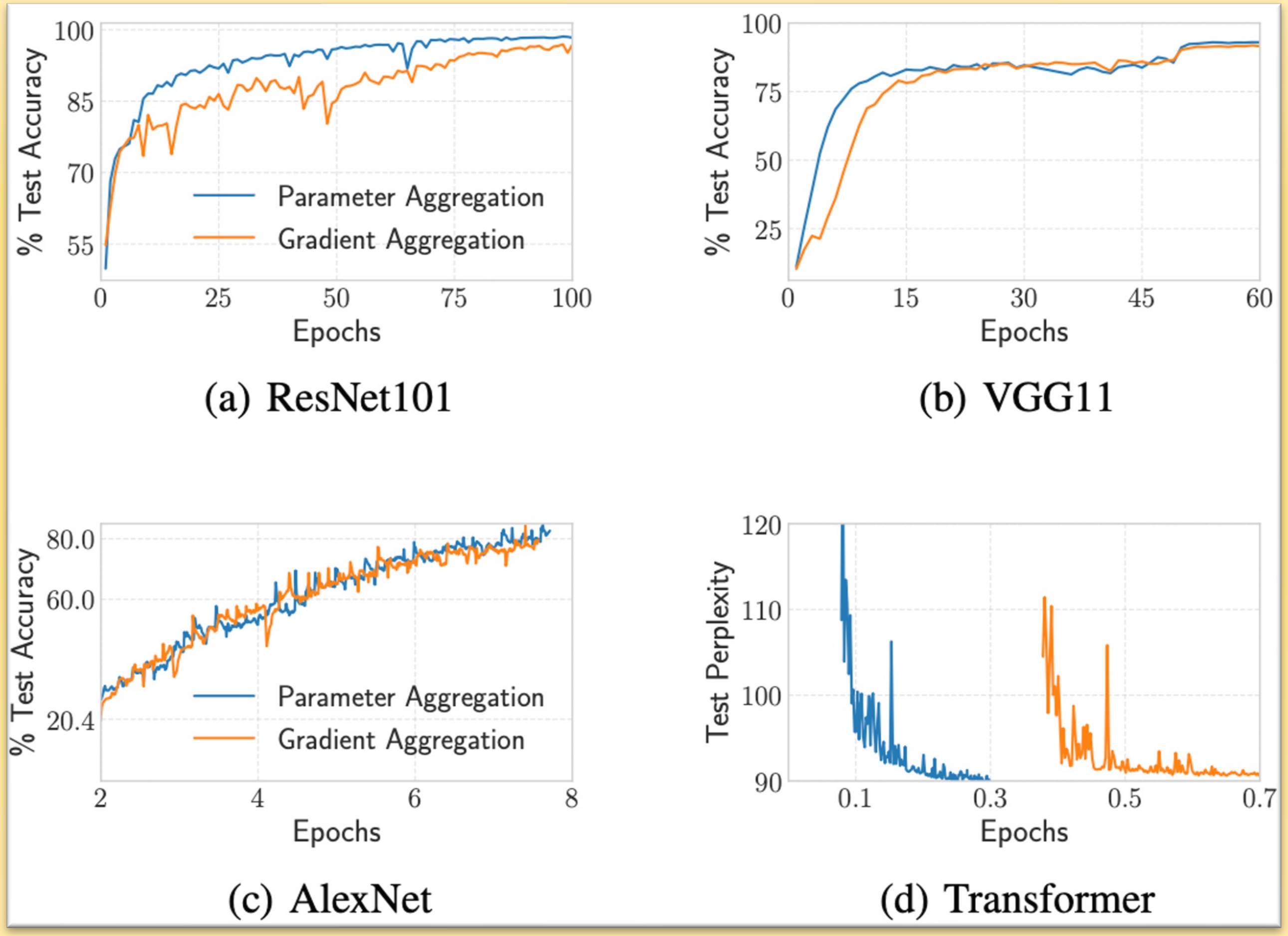
Assuming all workers start with the same model state



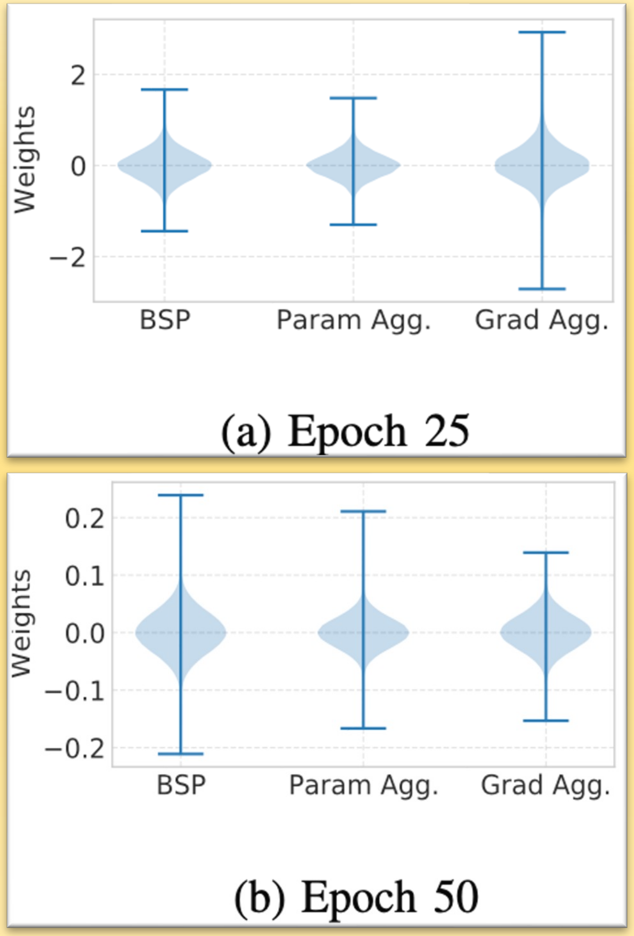
With gradient aggregation in SelSync $w_{(i+1,1)} \neq w_{(i+1,n)}$

Gradient vs. Parameter Aggregation in SelSync

Set delta = 0.25 with SelDP scheme



ResNet101



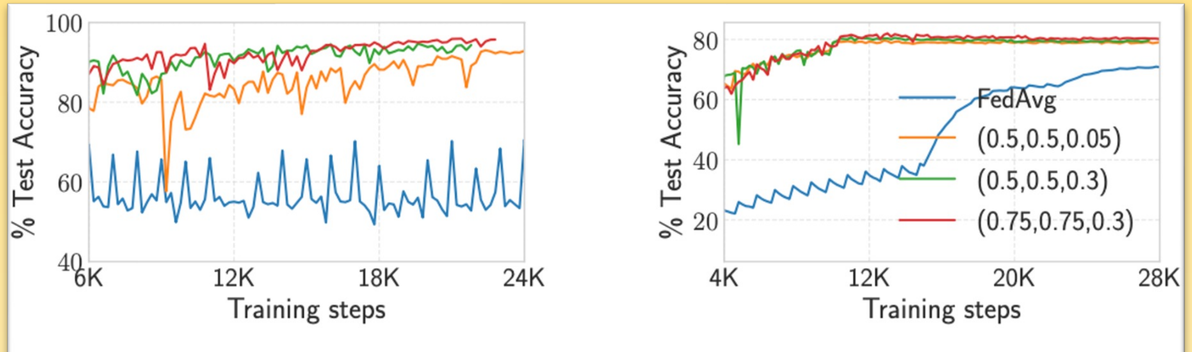
Training on unbalanced and Non-I.I.D. data

- Federated learning suffers from low convergence due to unbalanced and skewed data distribution
- ***Randomized Data-injection*** **[9]** improves distribution while preserving privacy
- Random subset of workers share partial training data at each iteration with (alpha, beta) params
- **However, batch-size is a sensitive hyperparameter that affects final model quality**

$$b' = \frac{b}{(1 + \alpha\beta N)}$$

Data-injection in SelSync needs (alpha, beta, delta) config

SelSync vs. FedAvg with non-I.I.D. data



(a) ResNet101 on CIFAR10 (b) VGG11 on CIFAR100

Implementation and Evaluation

Algorithm 1: {**Sel**}ective {**Sync**}hronization

Input: learning rate η, gradient change threshold δ, cluster-size N, training data $\mathcal{D}_n$ on worker with id n

procedure `train()`:

- $w_{(n,0)}$ = `pullFromPS()` ▷ initialize parameters
- **for** *i=0,1,...I on worker id n* ▷ *training iterations*
  - `bit` [N] `flags` = 0 ▷ synchronization status
  - $d_{(i,n)} \in \mathcal{D}_n$ ▷ sample mini-batch from data
  - $g_i = \nabla \mathcal{F}(x_{(i,k)}, w_{(n,i)})$ ▷ compute gradient at i
  - $\triangle(g_i)$ = `RelativeGradChange` $(||g_i||^2)$
  - $w_{(n,i+1)} = w_{(n,i)} - \eta \cdot g_i$ ▷ apply local updates
  - **if** $\triangle(g_i) \geq \delta$:
    - `flags` [n] = 1 ▷ synchronize called by worker n as its gradient change exceeds δ
  - `flags` = `allgather_status(flags)` ▷ call all-gather on `flags` such that index n holds worker n's synchronization status bit
  - **if** $1 \in$ `flags` :
    - `pushToPS` $(w_{(n,i+1)})$ ▷ push local updates
    - $w_{(n,i+1)}$ = `pullFromPS()` ▷ pull global

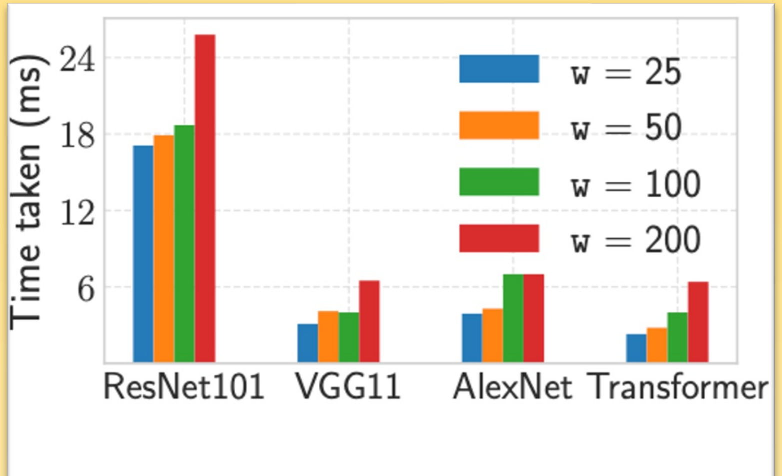
- Implemented in PyTorch over PS architecture
- Tested on a 16 V100 GPU cluster for IID data, 10-nodes for non-IID data
- Models trained: ResNet101, VGG11, AlexNet, Transformer
- Datasets used: CIFAR10, CIFAR100, ImageNet, WikiText-103

We compare SelSync with BSP, FedAvg and SSP

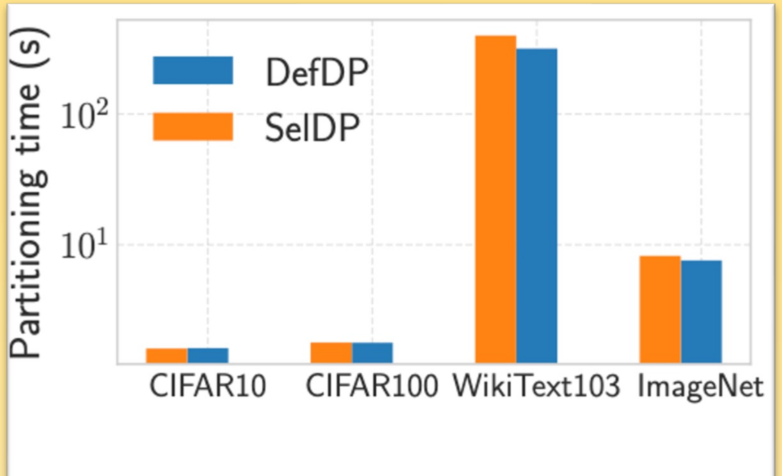
Metrics: Final accuracy/perplexity, overall speedup over BSP

SelSync Overheads

- Gradients computed over each iteration can be noisy; smoothing applied on ***Relative Gradient Change***
- Additional overhead of partitioning training data with ***SelDP*** scheme



(a) $\triangle(g_i)$ computation overhead



(b) Data-partitioning overhead

Training Performance

FA1: FedAvg (C, E) = (1, 0.25)

FA2: FedAvg (C, E) = (1, 0.125)

FA3: FedAvg (C, E) = (0.5, 0.25)

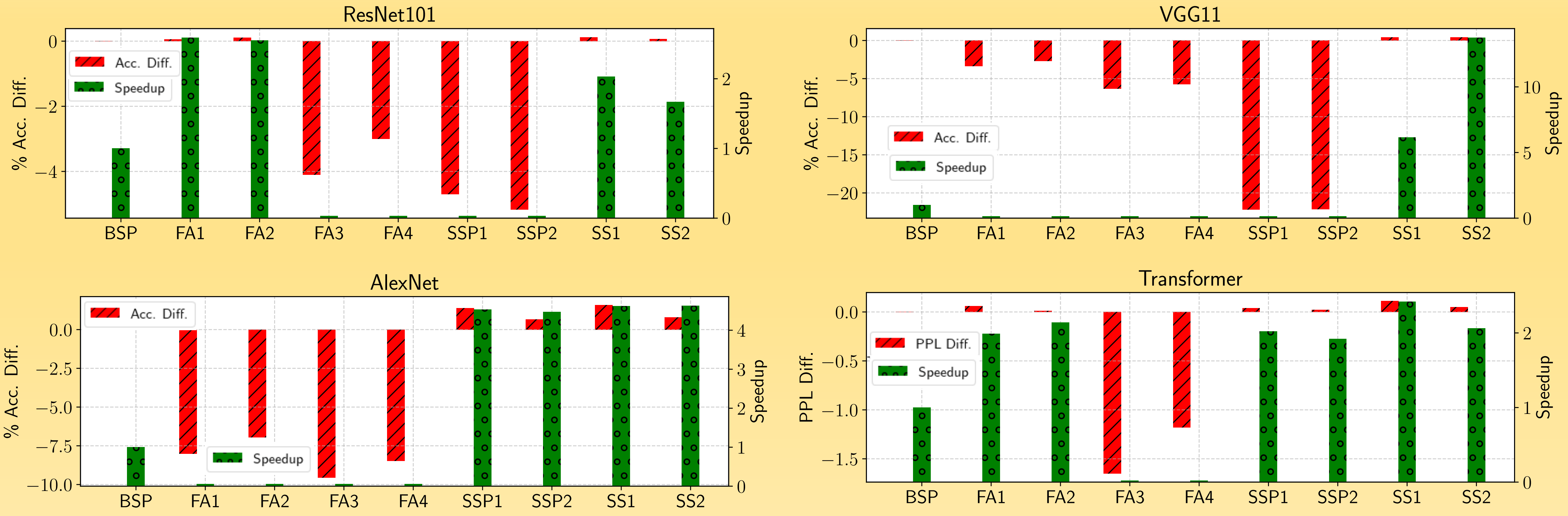
FA4: FedAvg (C, E) = (0.5, 0.125)

SSP1: SSP s = 100

SSP2: SSP s = 200

SS1: SelSync delta = 0.3

SS2: SelSync delta = 0.5



Related Work

Parallel and Statistical efficiency in distributed training:

[1] Scavenger: A Cloud Service for Optimizing Cost and Performance of ML Training

[2] GraVAC: Adaptive Compression for Communication-Efficient Distributed DL Training

Sensitive/Critical Regions in DNN Training:

[3] The Early Phase of Neural Network Training

[4] Critical Learning Periods in Deep Neural Networks

[5] Accordion: Adaptive Gradient Compression via Critical Learning Regime Identification

Related techniques/methods:

[6] BSP (on PS): Scaling Distributed Machine Learning with the Parameter Server

[7] FedAvg: Communication-Efficient Learning of Deep Network from Decentralized Data

[8] SSP: More Effective Distributed ML via a Stale-Synchronous Parameter Server

[9] ScaDLES: Scalable Deep Learning over Streaming Data at the Edge

Conclusion

- ***Relative Gradient Change*** serves as an effective indicator of measuring the significance of each gradient update in DNN training
- ***BSP*** has high synchronization cost; ***FedAvg*** mitigates communication with infrequent aggregation but degrades model generalization; training with ***SSP*** saturates convergence due to stale updates
- ***SelSync*** achieves similar accuracy to BSP while reducing communication depending on delta value. Speeds up training by up to 14x in our evaluation
- Large delta raises the threshold for communication, prioritizing speedup over convergence. Small delta increases synchronization frequency and favors convergence quality.
- Randomized data-injection is effective in the context of semi-synchronous training when training data is skewed and unbalanced

Thank you!

Accelerating Distributed ML Training via Selective Synchronization

https://sahiltyagi.academicwebsite.com/

Lossy Compression

Distributed Systems

Communication Optimization

Federated Learning

Large-scale ML

Differential Privacy

Cloud Computing

Adaptive Systems

Stream processing