

COMS E6998 010

Practical Deep Learning Systems Performance

Lecture 4 10/08/20

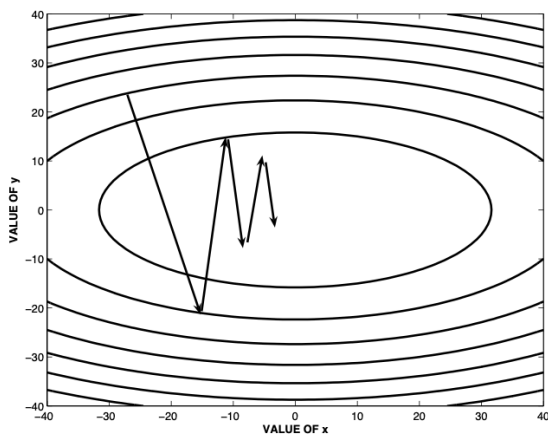
Logistics

- Homework 2: Posted Oct 2. Due Oct. 16 by 11:59 PM
- Prepare for Quiz; will be posted tomorrow
- Seminar sign-up due 10/16/2020
- Project proposals due 10/29/2020

Recall from Last lecture

- Single layer perceptron
- Non-linear activation functions and loss functions
- Multi layer neural networks; Hidden layers and their role
- Steps in training a deep neural network; Stochastic gradient descent
- Softmax activation function
- Vanishing gradient problem; Weight initialization and normalization
- Batch normalization
- Learning rate schedules
- Batch size and effective batch size
- Learning rate and batch size relationship

Gradient Descent Convergence



(b) Loss function is elliptical bowl

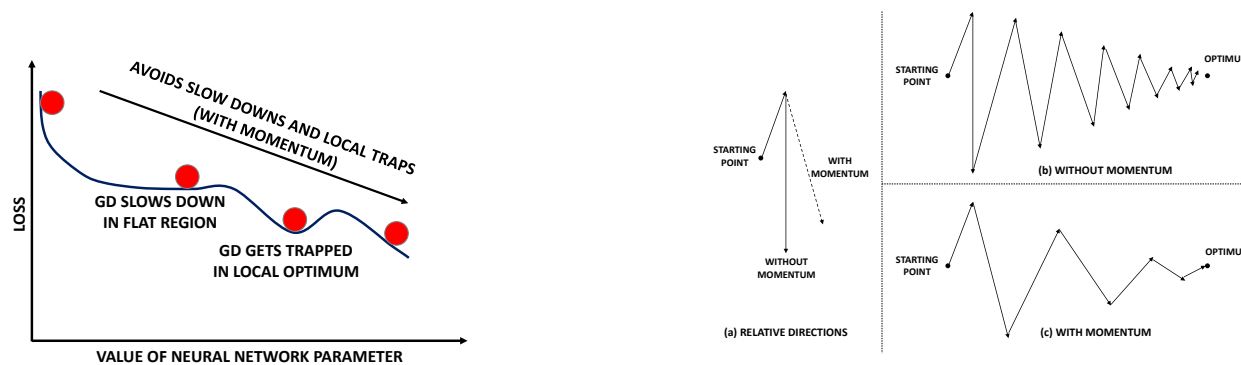
$$L = x^2 + 4y^2$$

We need to :

- Move quickly in directions with small but consistent (pointing in one direction, +ve or -ve) gradients.
- Move slowly in directions with big but inconsistent (oscillating between -ve and +ve) gradients.

- GD convergence is poor due to difference in gradient values along different dimensions
- Effective descent direction gets away from the minima if we use finite learning rate
- Gradient descent might also get trapped at saddle points and/or local minima

Gradient Descent with Momentum




- Add momentum to GD updates:

$$\overline{V} \leftarrow \beta \overline{V} - \alpha \frac{\partial L}{\partial \overline{W}}; \quad \overline{W} \leftarrow \overline{W} + \overline{V}$$

- Learning is accelerated as oscillations are damped and updates progress in the consistent directions of loss decrease
- Enables working with large learning rate values and hence faster convergence

Nesterov Momentum

- Simple momentum-based updates cause solution to overshoot the target minima
- Idea is to use some lookahead in computing the updates

$$\bar{V} \Leftarrow \underbrace{\beta \bar{V}}_{\text{Momentum}} - \alpha \frac{\partial L(\bar{W} + \beta \bar{V})}{\partial \bar{W}}; \quad \bar{W} \Leftarrow \bar{W} + \bar{V}$$


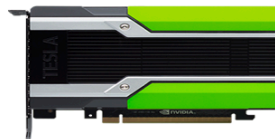
- Put on the brakes as the marble reaches near bottom of hill.
- Difference from standard momentum method in terms of where the gradient is computed.

Single Node, Single GPU Training

- Training throughput depends on:
 - Neural network model (activations, parameters, compute operations)
 - Batch size
 - Compute hardware: GPU type (e.g., Nvidia M60, K80, P100, V100)
 - Floating point precision (FP32 vs FP16)
 - Using FP16 can reduce training times and enable larger batch sizes/models without significantly impacting the accuracy of the trained model
- Increasing batch size increases throughput
 - Batch size is restricted by GPU memory
- Training time with single GPU very large: 6 days with Places dataset (2.5M images) using Alexnet on a single K40.
- Small batch size => noisier approximation of the gradient => lower learning rate => slower convergence

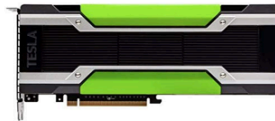
Commonly used GPU Accelerators in Deep Learning

Nvidia M60



2 GPUs with 4.8 TFLOPS SP
and 8 GB Cache each

Nvidia K80



2 GPUs with 3 TFLOPS SP
and 12 GB Cache each

Nvidia P100



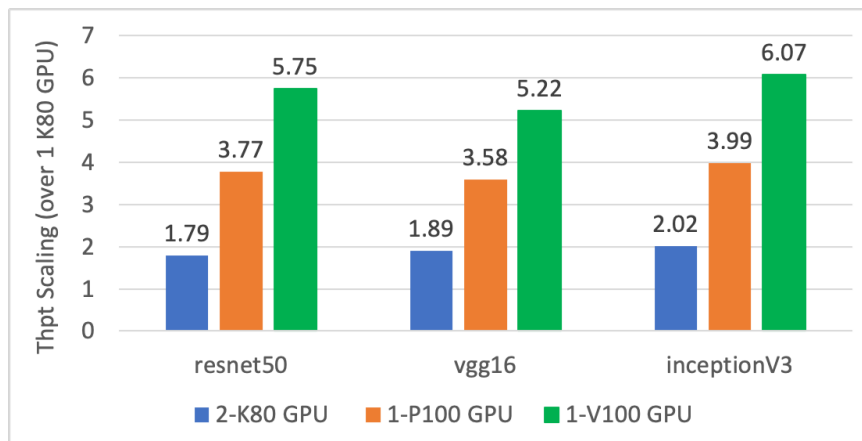
1 GPU with 9 TFLOPS SP
and 16 GB Cache

Nvidia V100



1 GPU with 14 TFLOPS SP
and 16 GB Cache

DL Scaling with GPU types: single GPU case

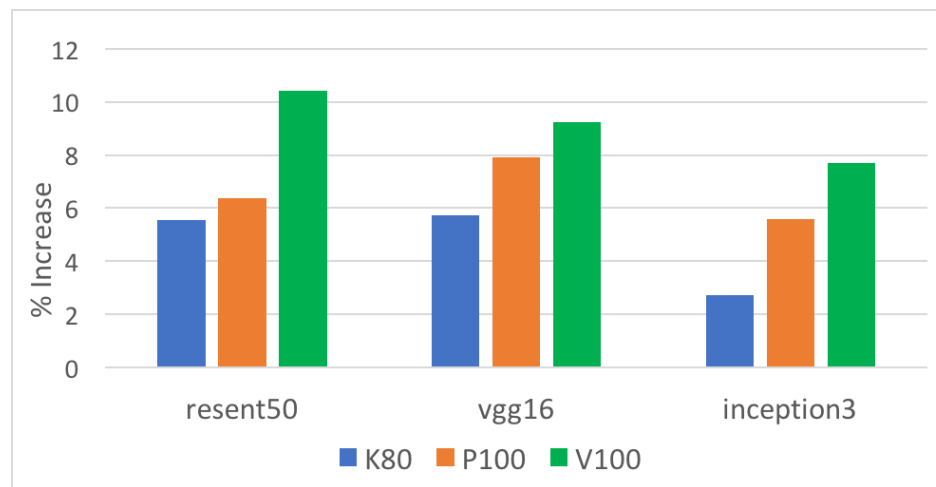


Training performance speed-up of DL models on different GPUs compared to 1-K80 GPU

GPU	Observed		
	Resnet50	Vgg16	InceptionV3
1-P100	2.11x	1.89x	1.98x
1-V100	3.21x	2.76x	3x

GPU scaling: Compared to K80 Acelerator (2 GPUs)

DL Scaling with Batch size on Single GPU



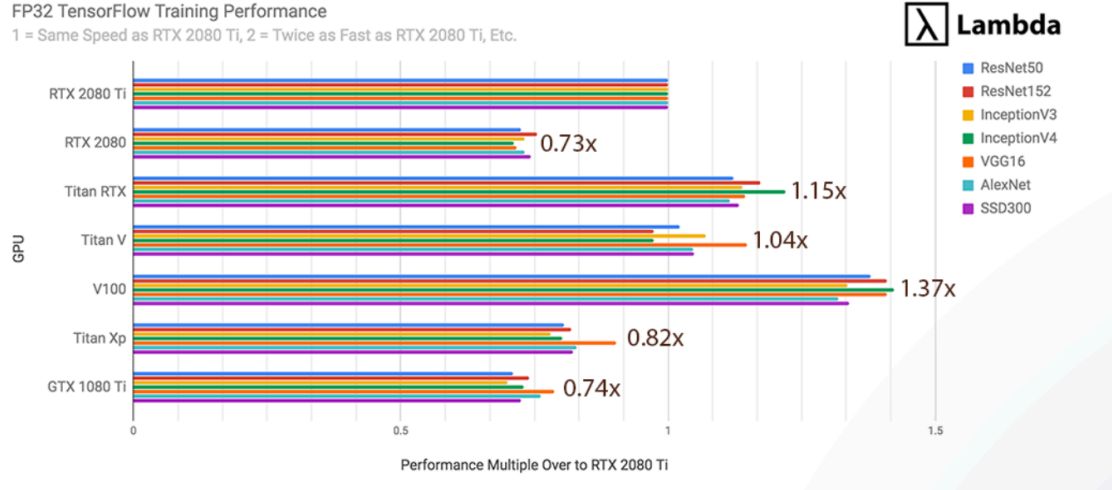
Training performance speed-up of DL models at batch size 64 compared to batch size 32.

- *Batch size scaling*: 10% for ResNet50 vs. 8% for InceptionV3 on a V100 GPU
- Speedup with increasing batch size is model dependent

Single GPU Training

FP32 TensorFlow Training Performance

1 = Same Speed as RTX 2080 Ti, 2 = Twice as Fast as RTX 2080 Ti, Etc.



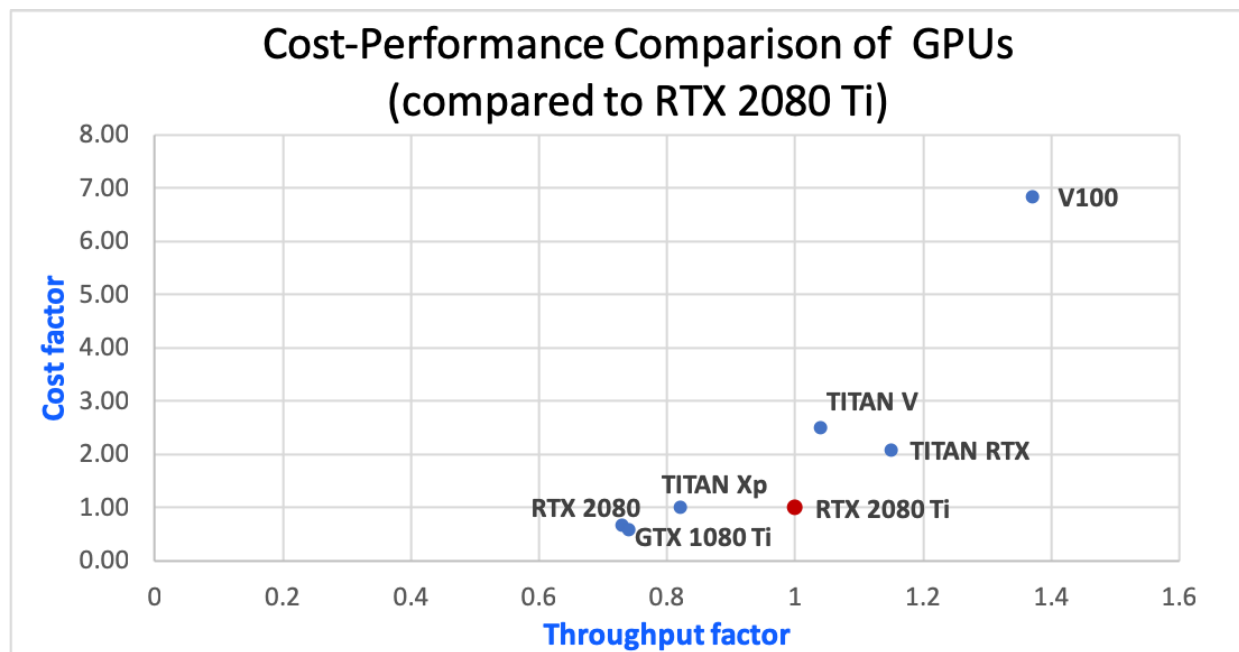
GPU Prices

- RTX 2080 Ti: \$1,199.00
- RTX 2080: \$799.00
- Titan RTX: \$2,499.00
- Titan V: \$2,999.00
- Tesla V100 (32 GB): ~\$8,200.00
- GTX 1080 Ti: \$699.00
- Titan Xp: \$1,200.00

The scaling with GPU type is dependent on neural network architecture.

<https://lambdalabs.com/blog/2080-ti-deep-learning-benchmarks/>

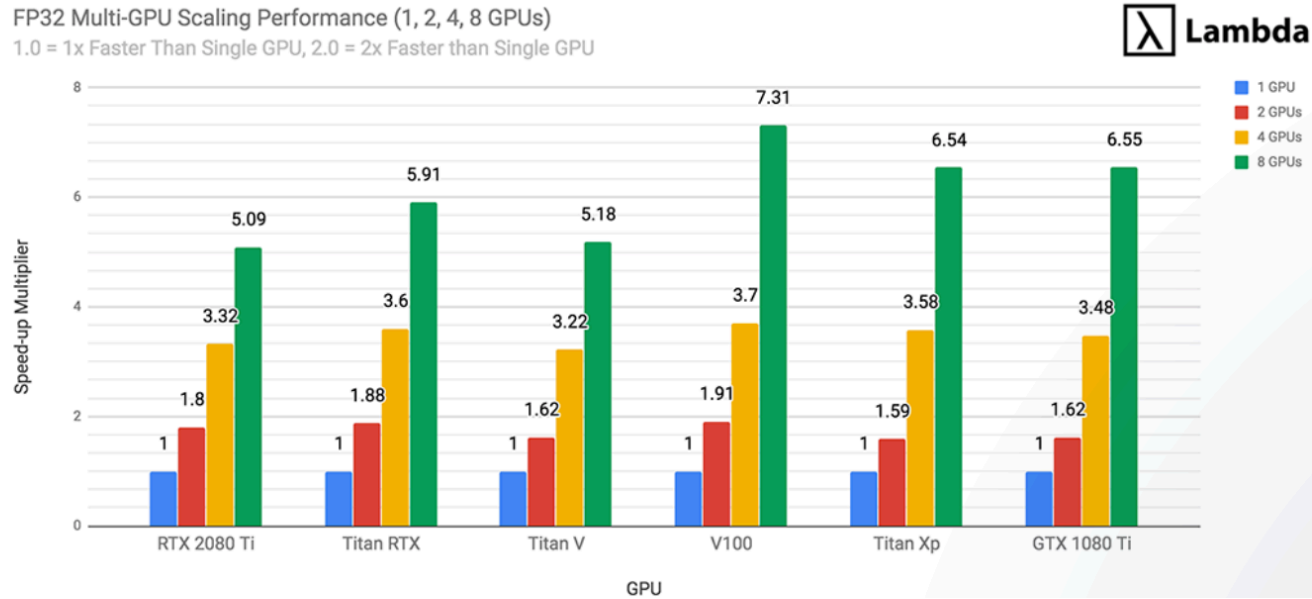
Cost-Performance Tradeoff



Points lying above the straight line connecting RTX 2080 Ti and origin corresponds to GPUs which are costlier than RTX 2080 Ti but don't offer same proportional increase in throughput.

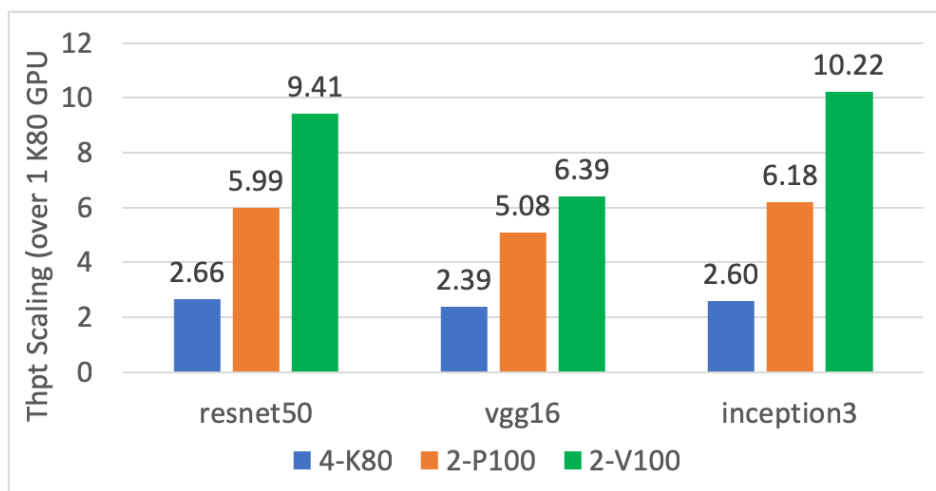
Single Node, Multi-GPU Training

FP32 Multi-GPU Scaling Performance (1, 2, 4, 8 GPUs)
1.0 = 1x Faster Than Single GPU, 2.0 = 2x Faster than Single GPU



<https://lambdalabs.com/blog/2080-ti-deep-learning-benchmarks/>

DL Scaling with GPU types: multi GPU case



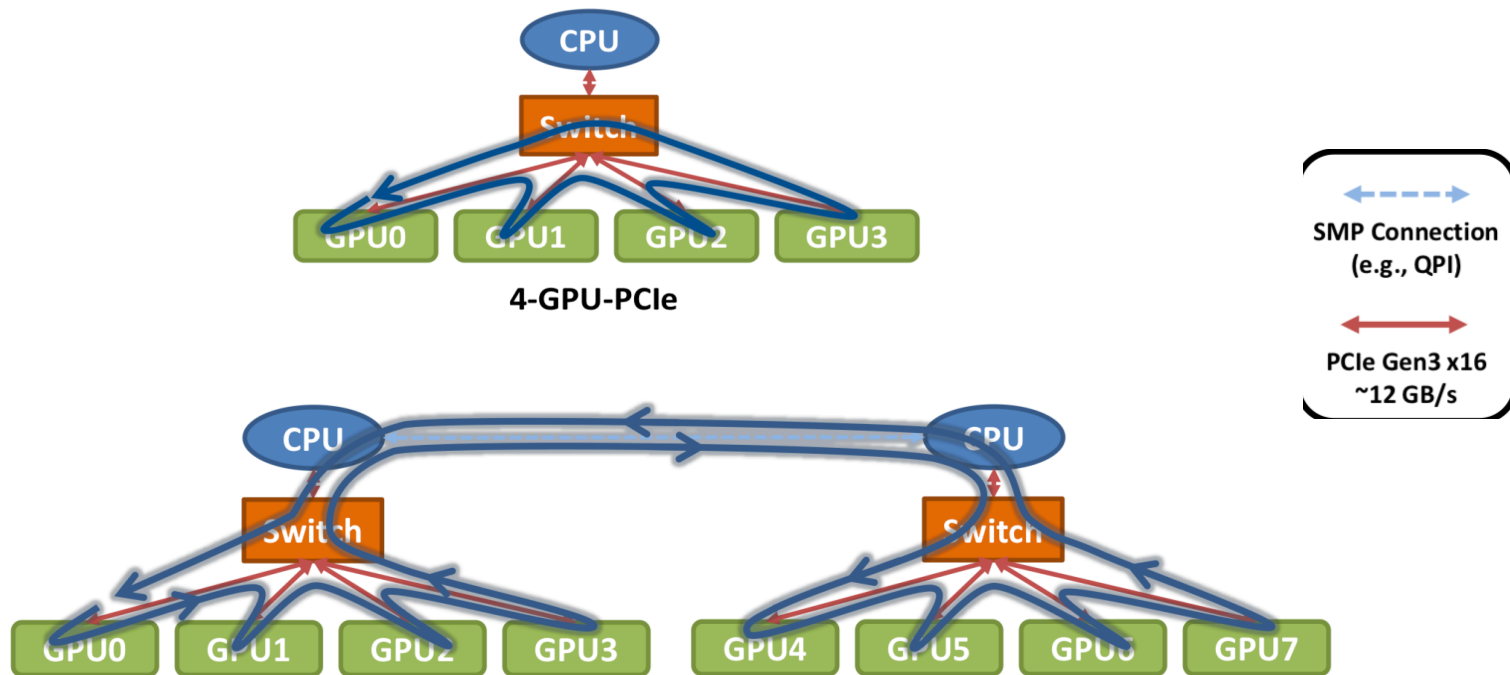
Training performance speed-up of DL models on different GPUs compared to 1-K80 GPU

- *GPU scaling*: ResNet50 and InceptionV3 achieve much higher speed-up (~6x with 2 P100s and ~10x with 2 V100s) compared to VGG16 (~5x with 2 P100s and ~6x with 2 V100s)

Single Node, Multi GPU Training

- Communication libraries (e.g., NCCL) and supported communication algorithms/collectives (broadcast, all-reduce, gather)
 - NCCL (“Nickel”) is library of accelerated collectives that is easily integrated and topology-aware so as to improve the scalability of multi-GPU applications
- Communication link bandwidth: PCIe/QPI or NVlink
- Communication algorithms depend on the communication topology (ring, hub-spoke, fully connected) between the GPUs.
- Most collectives amenable to bandwidth-optimal implementation on rings, and many topologies can be interpreted as one or more rings [P. Patarasuk and X. Yuan]

Ring based collectives



PCIe and NVLink

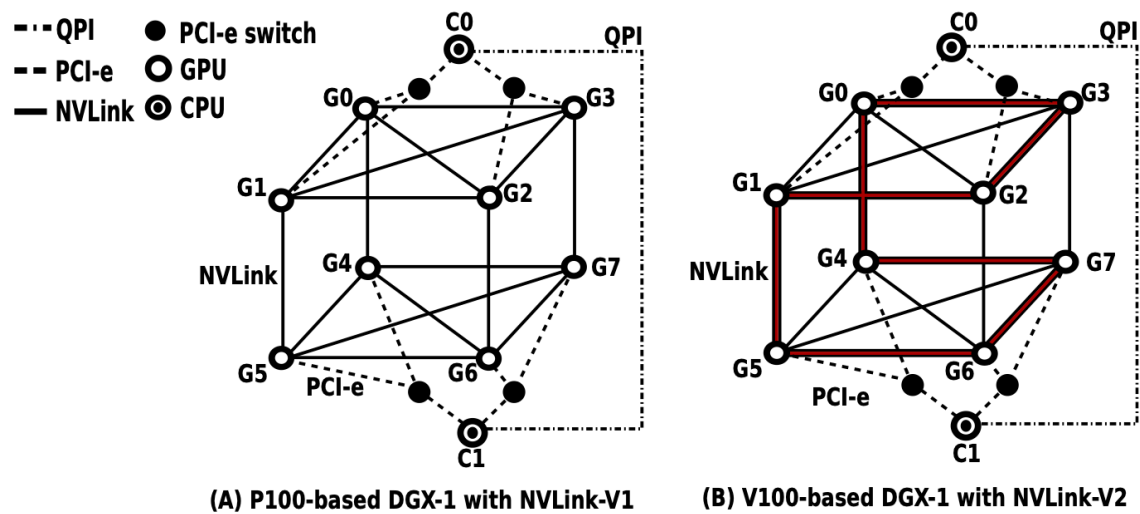


Fig. 1: PCIe and NVLink-V1/V2 topology for P100-DGX-1 and V100-DGX-1.

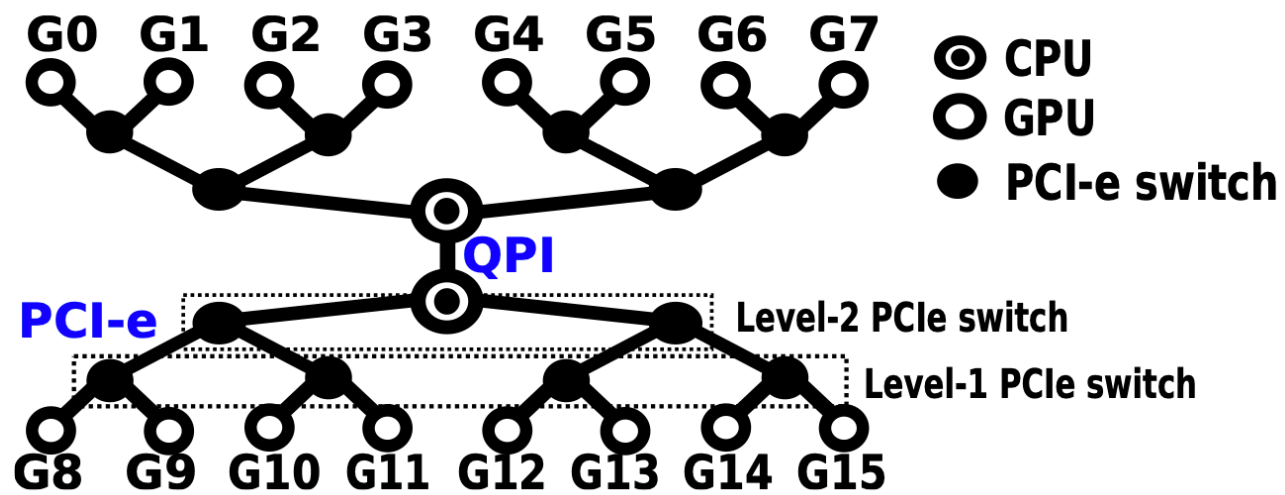
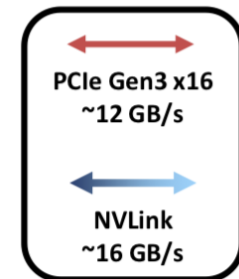
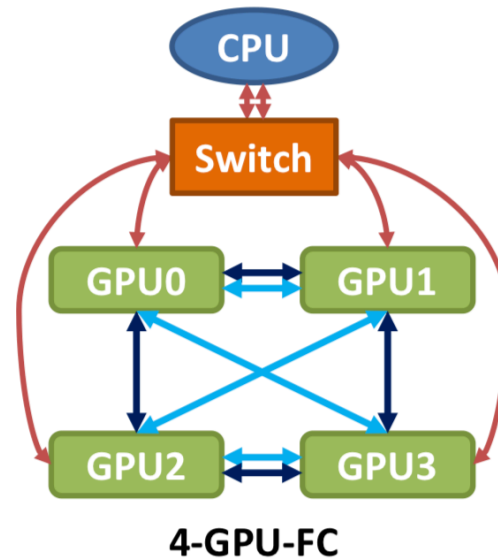
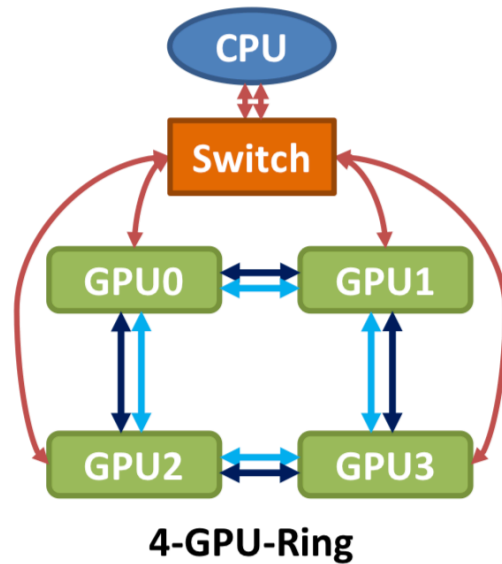


Fig. 4: PCIe interconnect topology in DGX-2.

Ring based collectives with NVLink



Distributed Training

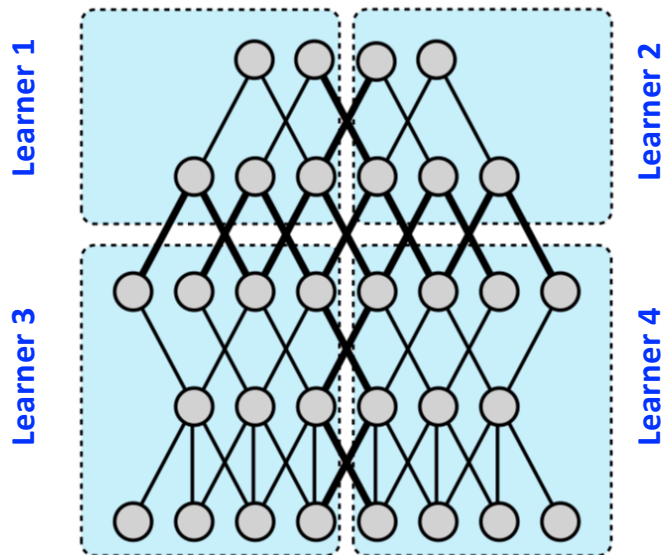
- Type of Parallelism: Model, Data, Hybrid
- Type of Aggregation: Centralized, decentralized
- Centralized aggregation: parameter server
- Decentralized aggregation: P2P, all reduce
- Performance metric: Scaling efficiency
 - Defined as the ratio between the run time of one iteration on a single GPU and the run time of one iteration when distributed over n GPUs.

Parallelism

- Parallel execution of a training job on different compute units through scale-up (single node, multiple and faster GPUs) or scale-out (multiple nodes distributed training)
- Enables working with large models by partitioning model across learners
- Enables efficient training with large datasets using large “effective” batch sizes (batch split across learners)
 - Speeds up computation
- Model, Data, Hybrid Parallelism

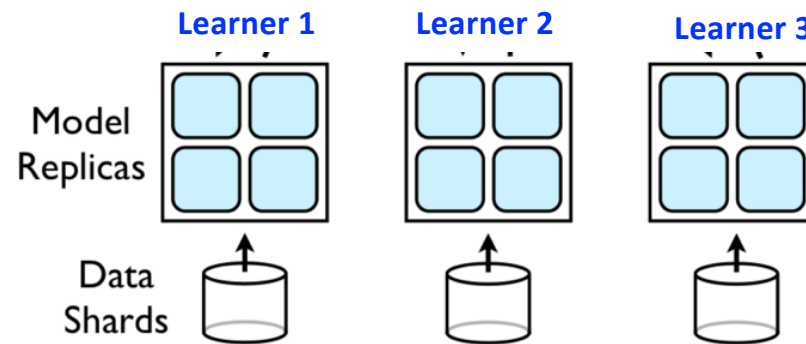
Model Parallelism

- Splitting the model across multiple learners



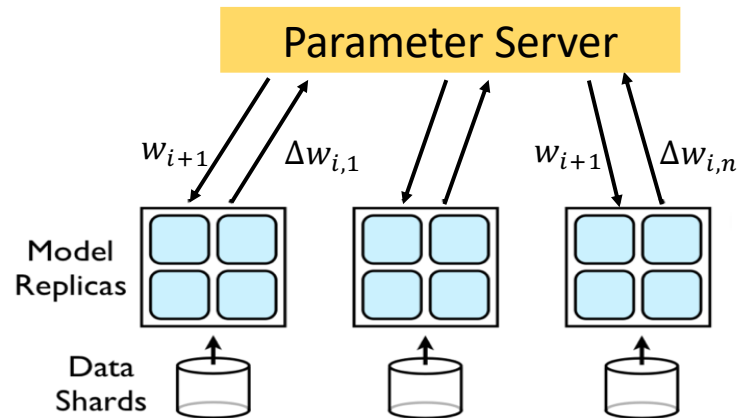
- 5 layered neural network
- Partitioned across 4 learners
- Bold edges cross learn boundaries and involve inter-learner communication
- Performance benefits depend on
 - Connectivity structure
 - Compute demand of operations
- Heavy compute and local connectivity –benefit most
 - Each machine handles a subset of computation
 - Low network traffic

Data Parallelism



- Model is replicated on different learners
- Data is sharded and each learner work on a different partition
- Helps in efficient training with large amount of data
- Parameters (weights, biases, gradients) from different replicas need to be synchronized

Parameter server (PS) based Synchronization

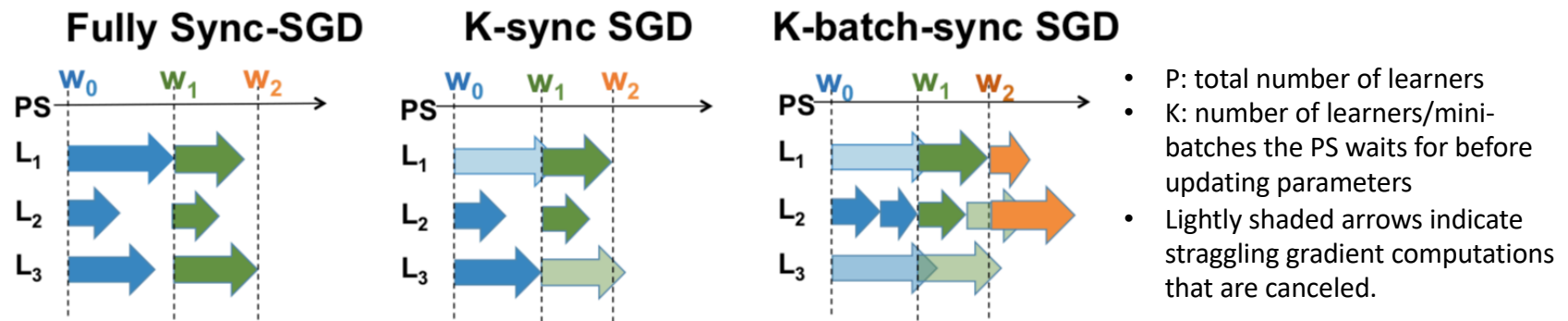


- Each learner executes the entire model
- After each mini-batch processing a learner calculates the gradient and sends it to the parameter server
- The parameter server calculates new value of weights and sends them to the model replicas

Synchronous SGD and the Straggler problem

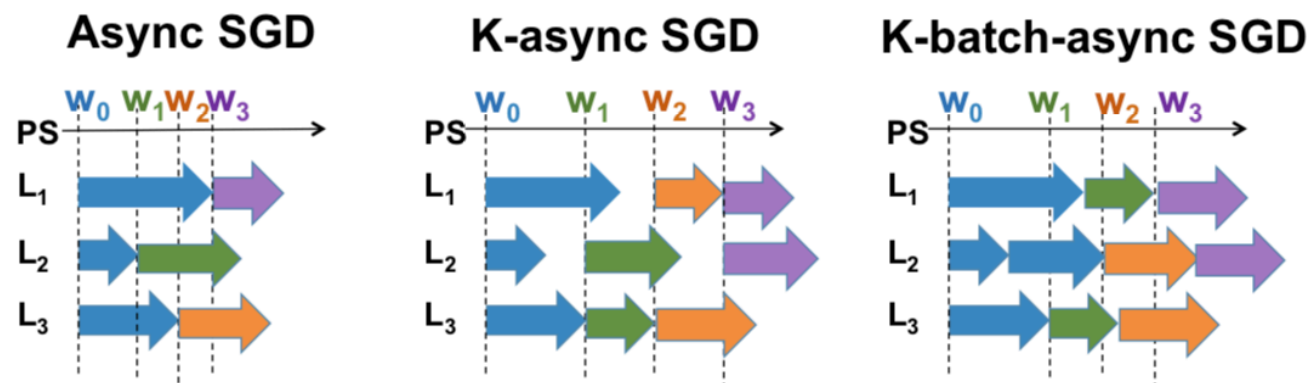
- PS needs to wait for updated gradients from all the learners before calculating the model parameters
- Even though size of mini-batch processed by each learner is same, updates from different learners may be available at different times at the PS
 - Randomness in compute time at learners
 - Randomness in communication time between learners and PS
- Waiting for slow and straggling learners diminishes the speed-up offered by parallelizing the training

Synchronous SGD Variants



- **K-sync SGD:** PS waits for gradients from K learners before updating parameters; the remaining learners are canceled
- When $K = P$, K-sync SGD is same as Fully Sync-SGD
- **K-batch sync:** PS waits for gradients from K mini-batches before updating parameters; the remaining learners are canceled
 - Irrespective of which learner the gradients come from
 - Wherever any learner finishes, it pushes its gradient to the PS, fetches current parameter at PS and starts computing gradient on the next mini-batch based on the same local value of the parameters
- Runtime per iteration reduces with K-batch sync; error convergence is same as K-sync

Asynchronous SGD and Variants

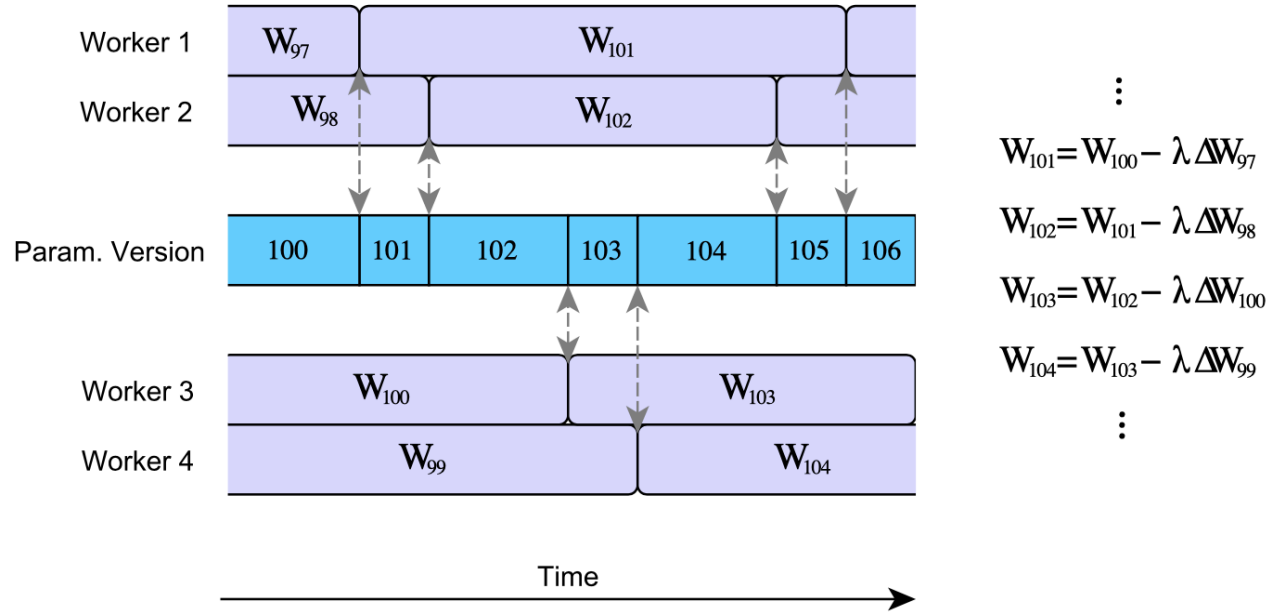


- **K-async SGD:** PS waits for gradients from K learners before updating parameters but the remaining learners are **not** canceled; each learner may be giving a gradient calculated at stale version of the parameters
- When $K = 1$, K-async SGD is same as Async-SGD
- **K-batch async:** PS waits for gradients from K mini-batches before updating parameters; **the remaining learners are not canceled**
 - Wherever any learner finishes, it pushes its gradient to the PS, fetches current parameter at PS and starts computing gradient on the next mini-batch based on the current value of the PS
- Runtime per iteration reduces with K-batch sync; error convergence is same as K-async

Asynchronous SGD and Stale Gradients

- PS updates happen without waiting for all learners
- Weights that a learner uses to evaluate gradients may be old values of the parameters at PS
 - Parameter server asynchronously updates weights
 - By the time learner gets back to the PS to submit gradient, the weights may have already been updated at the PS (by other learners)
 - Gradients returned by this learner are stale (i.e., were evaluated at an older version of the model)
- Stale gradients can make SGD unstable, slowdown convergence, cause sub-optimal convergence (compared to Sync-SGD)

Stale Gradient Problem in Async-SGD



$$\vdots$$

$$W_{101} = W_{100} - \lambda \Delta W_{97}$$

$$W_{102} = W_{101} - \lambda \Delta W_{98}$$

$$W_{103} = W_{102} - \lambda \Delta W_{100}$$

$$W_{104} = W_{103} - \lambda \Delta W_{99}$$

$$\vdots$$

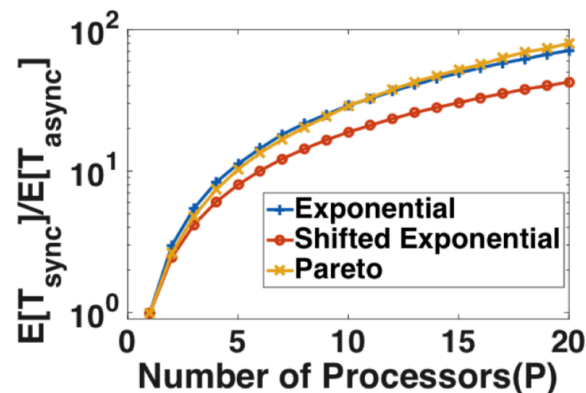
Slowdown of Sync-SGD: Analytical result

Theorem 1. Let the wall clock time of each learner to process a single mini-batch be i.i.d. random variables X_1, X_2, \dots, X_P . Then the ratio of the expected runtimes per iteration for synchronous and asynchronous SGD is

$$\frac{\mathbb{E}[T_{Sync}]}{\mathbb{E}[T_{Async}]} = P \frac{\mathbb{E}[X_{P:P}]}{\mathbb{E}[X]}$$

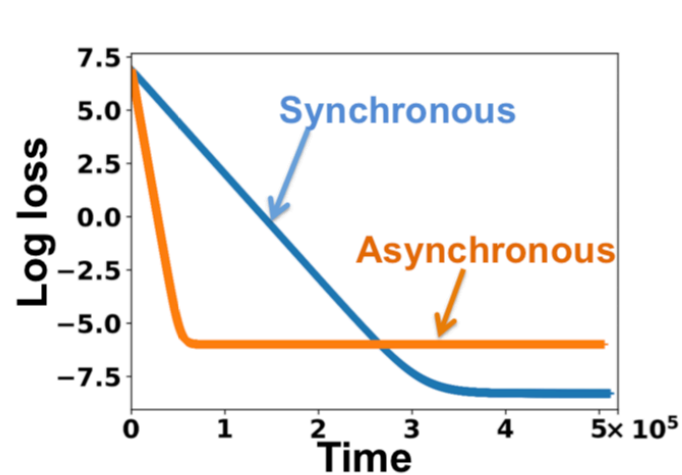
where $X_{(P:P)}$ is the P^{th} order statistic of P i.i.d. random variables X_1, X_2, \dots, X_P .

- For $X_i \sim \exp(\mu)$ $\frac{\mathbb{E}[T_{Sync}]}{\mathbb{E}[T_{Async}]} \sim P \log P$

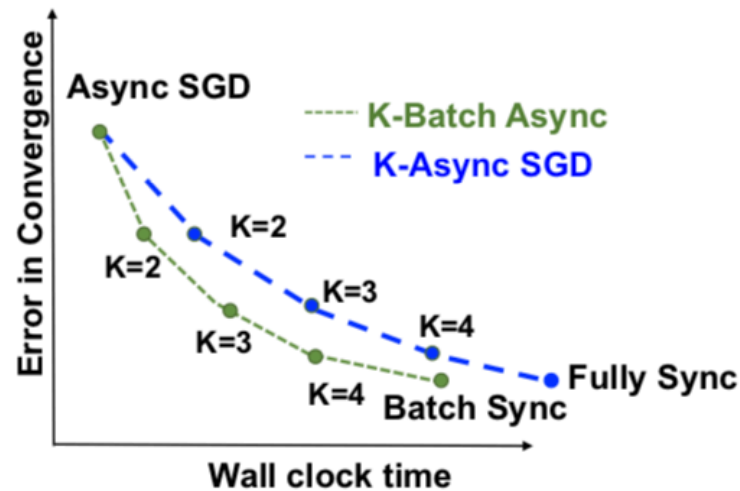


Dutta et al. Slow and stale gradients can win the race: error-runtime tradeoffs in distributed SGD. 2018

Error-Runtime Tradeoff in SGD Variants

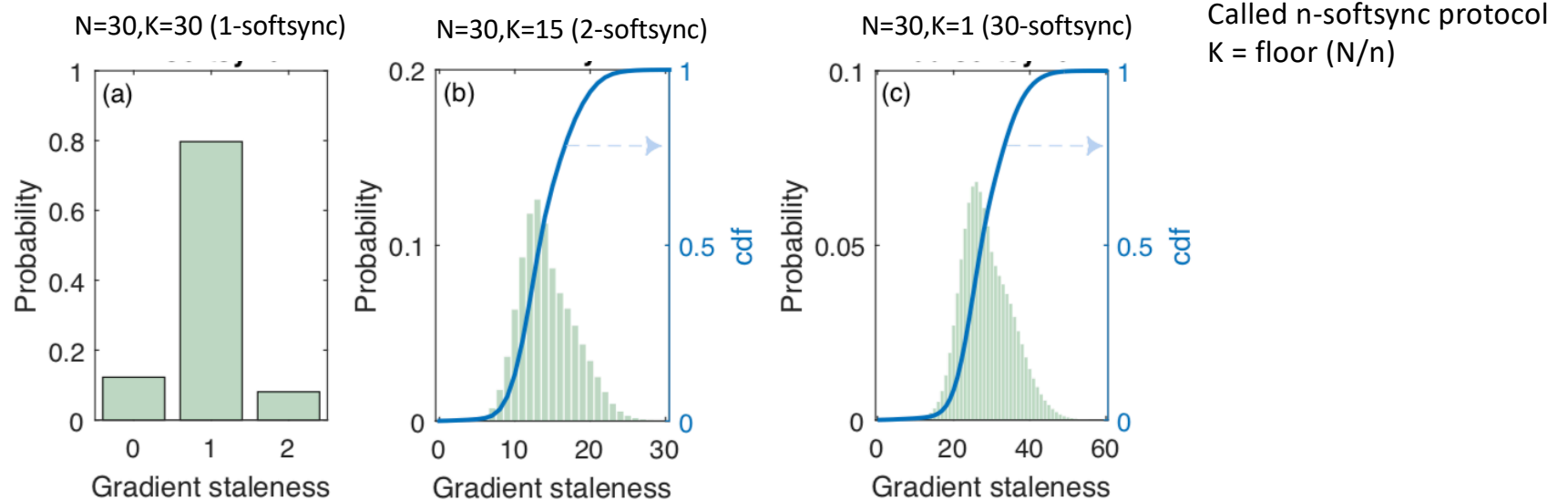


- Error-runtime trade-off for Sync and Async-SGD with same learning rate.
- Async-SGD has faster decay with time but a higher error floor.



Dutta et al. Slow and stale gradients can win the race: error-runtime tradeoffs in distributed SGD. 2018

Distribution of Staleness for K-batch Async

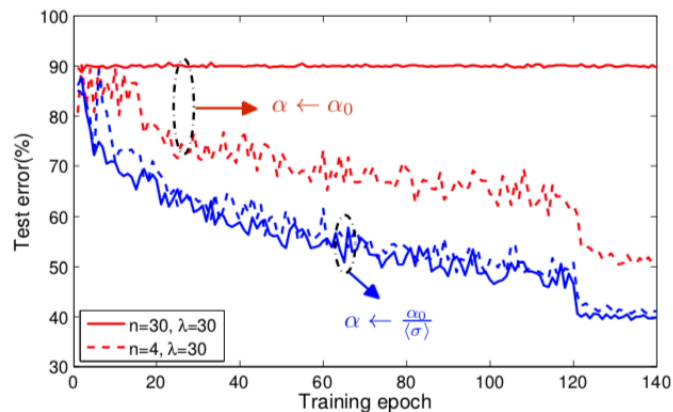


The gradient is on average N/K (N : number of learners) steps out of date by the time they are applied to the global parameter vector.

Staleness Dependent Learning Rate -1

- With staleness-dependent learning rate setting Async-SGD can achieve accuracy comparable to Sync-SGD while achieving linear speedup of ASGD
- Decrease the learning rate by mean staleness

$$\text{learning rate } (\alpha) = \frac{\text{base learning rate } (\alpha_0)}{\text{average staleness } (\langle \sigma \rangle)}$$



Gupta et al. Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study 2016

Staleness Dependent Learning Rate -2

$$\eta_j = \min \left\{ \frac{C}{\|\mathbf{w}_j - \mathbf{w}_{\tau(j)}\|_2^2}, \eta_{max} \right\}$$

η_j : learning rate at j th iteration of parameters at PS

\mathbf{w}_j : parameter value at j th iteration at PS

$\mathbf{w}_{\tau(j)}$: parameter value used by the learner (to calculate gradient)
whose gradient pushing triggered j th iteration at PS

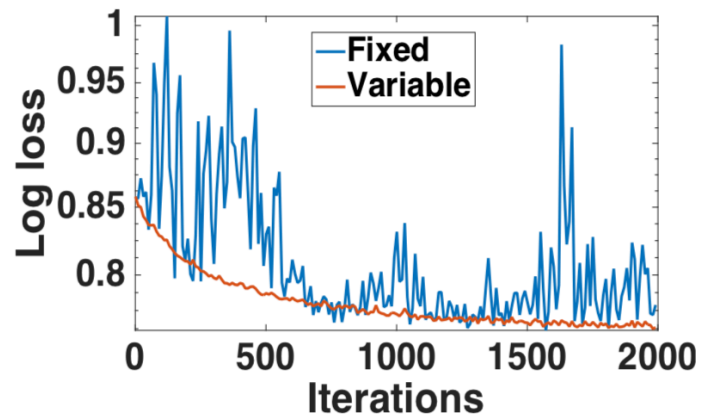
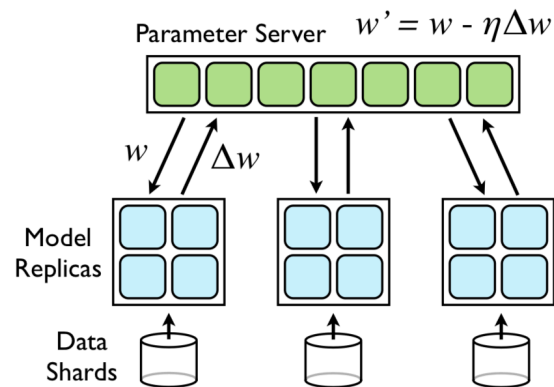


Figure 9: Async-SGD on CIFAR10 dataset, with $X \sim \exp 20$, mini-batch size $m = 250$ and $P = 40$ learners. We compare fixed $\eta = 0.01$, and the variable schedule given in (13) for $\eta_{max} = 0.01$ and $C = 0.005\eta_{max}$. Observe that the proposed schedule can give fast convergence, and also maintain stability, while the fixed η algorithm becomes unstable.

Dutta et al. Slow and stale gradients can win the race: error-runtime tradeoffs in distributed SGD. 2018

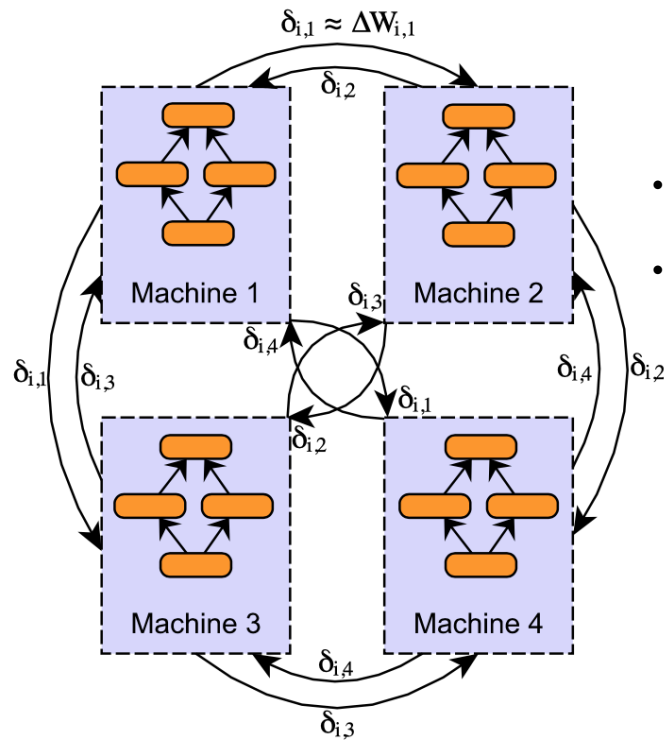
Sharded Parameter server



- Downpour SGD (Google, 2012)
- Asynchronous SGD
- Model and Data parallelism
- Parameter server is sharded

- Each PS shard responsible for updating portion of parameters; learners only send corresponding gradients to the PS shards
- PS sharding alleviates congestion at a single PS
- Different PS shards update their parameters asynchronously, they do not need to communicate among themselves
- Need to tune ratio of parameter servers to workers

Decentralized Aggregation: Peer-to-Peer



- Peer to peer communication is used to transmit model updates between workers).
- Updates are heavily compressed, resulting in the size of network communications being reduced by some 3 orders of magnitude.

Hardware support for efficient distributed training

- NCCL 1.0, NCCL 2.0, Nvlink, Infiniband, RDMA
- Link to NVIDIA presentation:

<http://on-demand.gputechconf.com/gtc/2017/presentation/s7155-jeauegy-nccl.pdf>

Prepare for Lecture 5

- Get ready with your cloud setup; Homework 2 will involve use of GPUs
- Required reading papers on distributed training
- Start working on your project proposal:
 1. Form your team and brainstorm
 2. Discuss your project idea with me
 3. Create your project proposal
- Lecture 5 will cover more on distributed training and specialized DL architectures

Seminar Reading List

- **Distributed Training**

- Jeff Dean et al. [Large Scale Distributed Deep Networks](#) 2012
- Alexander Sergeev, Mike Del Balso. [Horovod: fast and easy distributed deep learning in TensorFlow](#) 2018
- Minsik Cho et al. [PowerAI DDL](#) 2017
- TAL BEN-NUN, TORSTEN HOEFLER. [Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis](#)
- Andrew Gibiansky. [Bringing HPC techniques to deep learning](#) 2017
- Goyal et al. [Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour](#) 2017
- Gupta et al. [Staleness-aware Async-SGD for Distributed Deep Learning](#) 2016
- Gupta et al. [Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study](#) 2016
- Nikko Storm. [Scalable distributed dnn training using commodity gpu cloud computing](#) 2015

Suggested Reading List

- Sanghamitra Dutta et al. [Slow and stale gradients can win the race: error-runtime tradeoffs in distributed SGD](#). 2018
- Sebastian Ruder. [An overview of gradient descent optimization algorithms](#). 2016
- Stephen Balaban. [Distributed Training: A Gentle Introduction](#)
- [NCCL: Accelerated Multi-GPUs Collective Communications](#)
- Li et al. [Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect](#)
- Vitaly Bushaev. [Stochastic Gradient Descent with momentum](#). 2017