Distributed DNN Training

Vaibhav Saxena

Learning and Reasoning Group,
IBM Research - India, New Delhi
vaibhavsaxena@in.ibm.com

March 23, 2021

Outline

Part 1 – Distributed DNN Training

- Why DNN Training ?
- DNN and DNN Training Recap
- DNN Training Parallelization Strategies
  - Data Parallel
  - Model Parallel

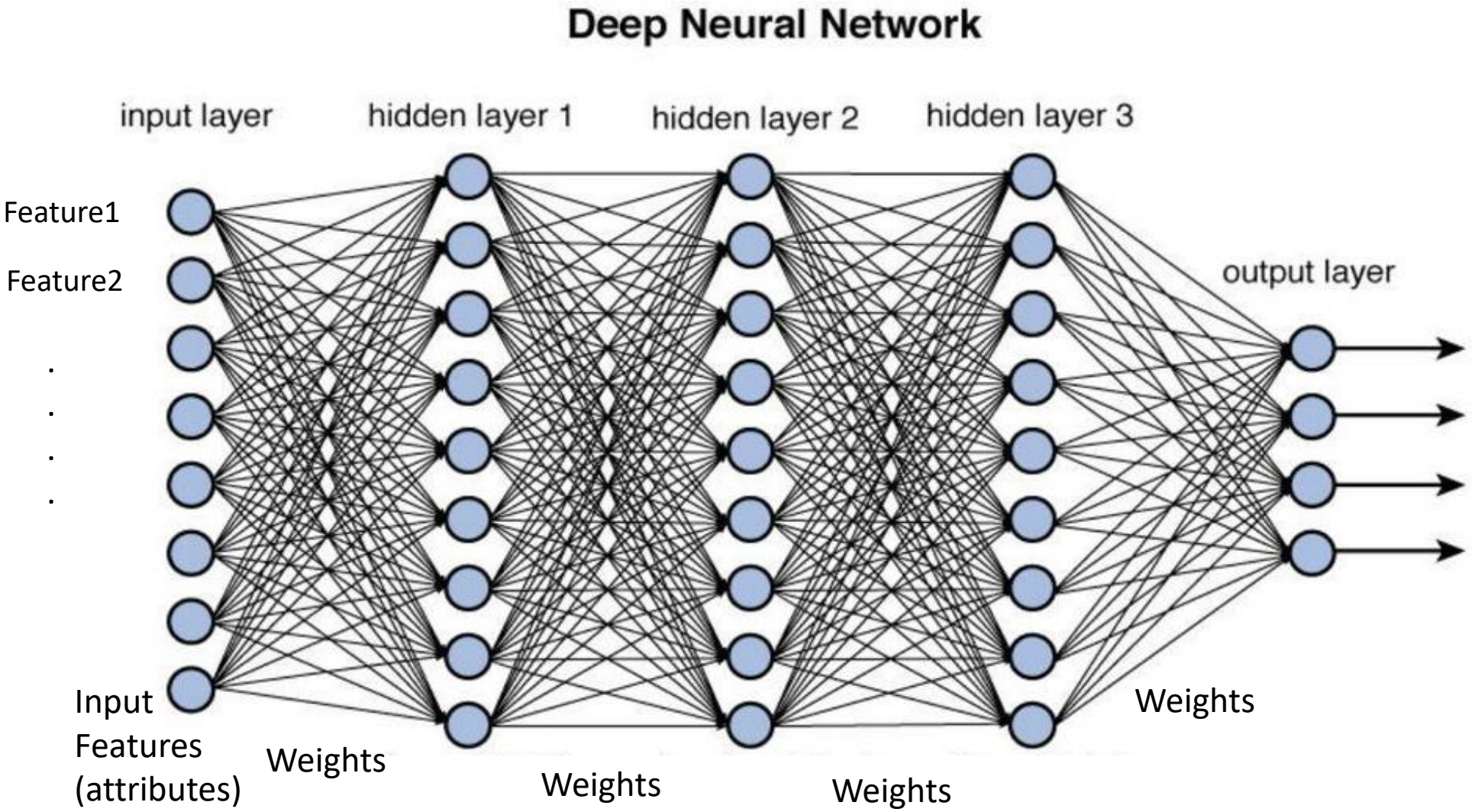
Part 2 - Large Scale DNN Training Example

- Training Imagenet-1K with Resnet-50
- Challenges and solutions

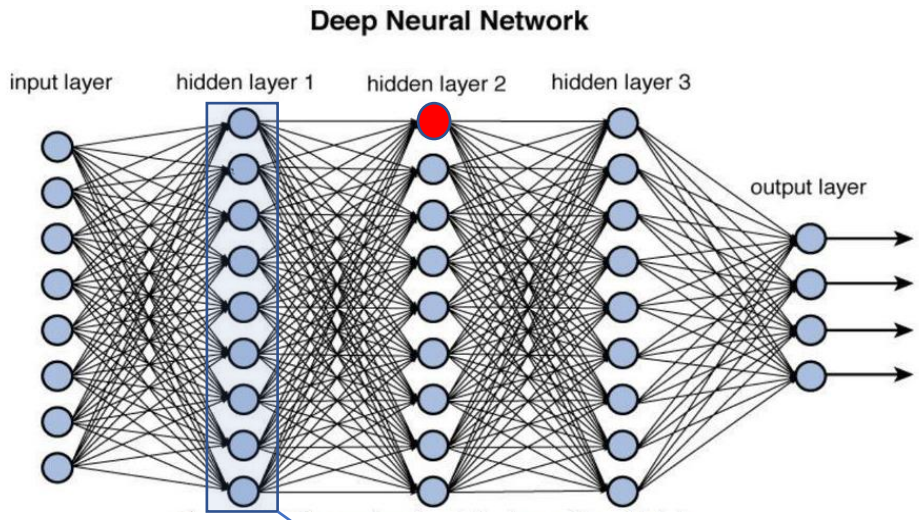
DNN Training: Challenges

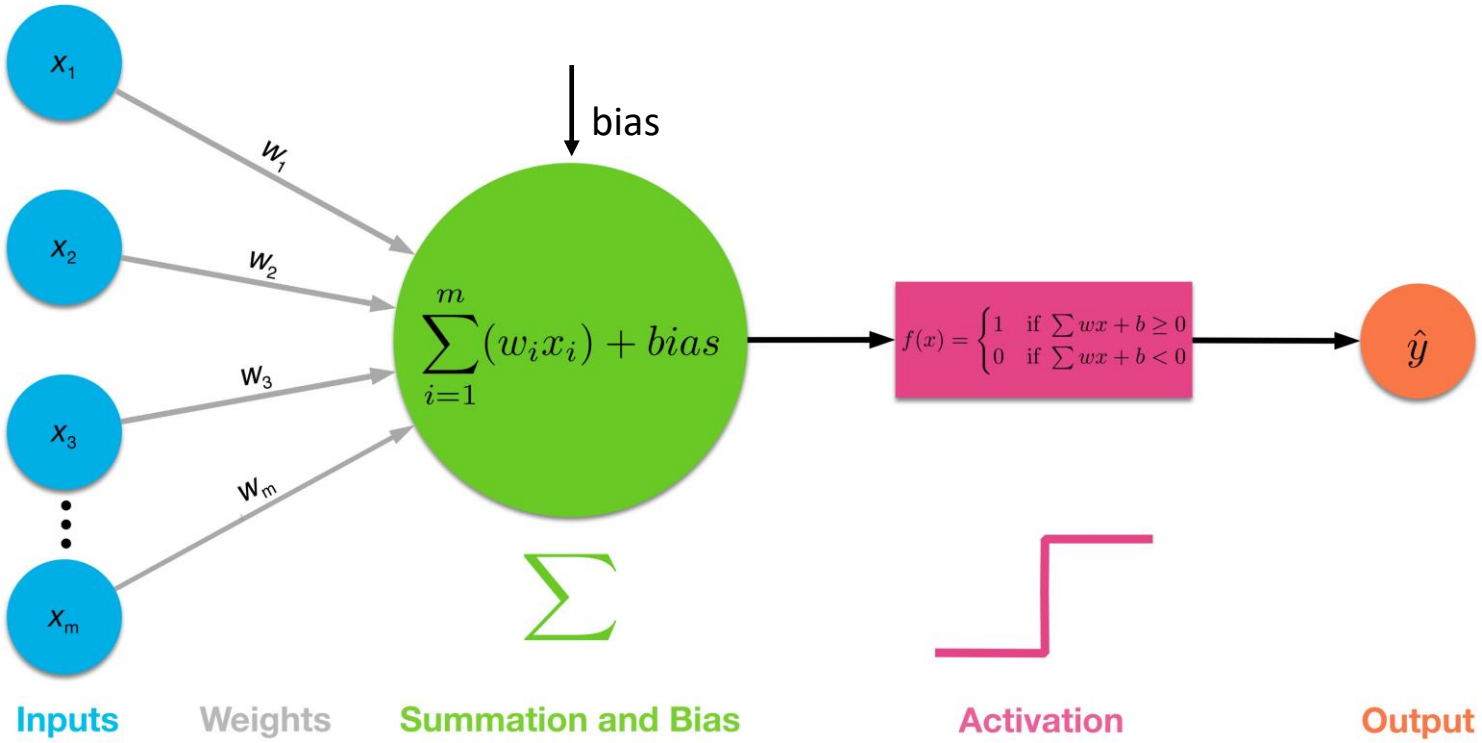
- One of the most data and compute intensive workloads around
- Can take several days even with GPUs
- Bigger data, more complex models -> More training time
- Need distributed DNN training

Deep Neural Network (DNN): Recap



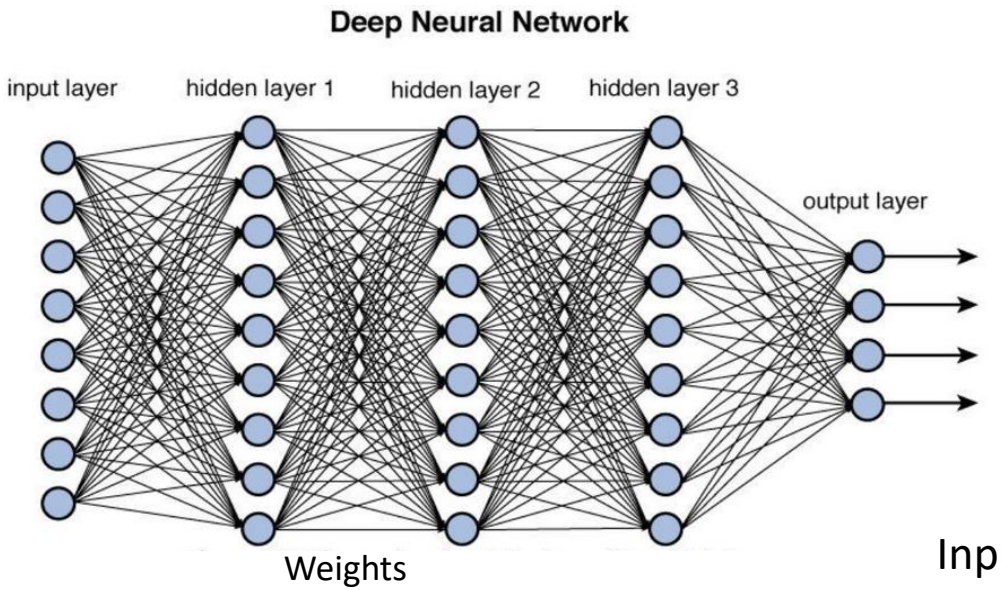
Deep Neural Network (DNN): Recap





DNN Training

Iteratively optimize model parameters (weights, bias) to accurately map inputs to outputs.



Training Steps

1. Forward Pass: Pass the input data through layers of DNN to get the output
2. Compute the Loss (e.g. cross entropy, MSE)
3. Backward Pass (Backpropagation): Back propagate through the layers to compute the gradients of loss wrt weights
4. Optimization Step: Update the weights through an optimizer (SGD)

Input – x, True output – y, Weights – $W = \{w_i : i = 1....n\}$

1. Forward Pass: $\hat{y} = FP(W, x)$
2. Loss $L = l(\hat{y}, y) = l(W, x, y)$
3. Backward Pass: Compute gradient $\partial L / \partial w_i$
4. Update Weights : $w_i = w_i - \alpha \, \partial L / \partial w_i$

DNN Training:
Mini-batch Stochastic Gradient Descent (SGD)

- **Epoch**: One pass of the full data through DNN
- **Mini-batch**: Fixed number of training samples from input data (64, 128…)
  - Number of Mini-batches – #Training Samples (N) / Mini-batch size (BS)

For epochs 1 to N_Epochs do

Shuffle the input data

For each epoch do

For each mini-batch do (N/BS batches)

1. Feed it to DNN (forward pass)
2. Compute Loss over the mini-batch
3. Calculate the mean gradient of the mini-batch (backward pass)
4. Use the mean gradient to update the weights (Optimization)

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \alpha \frac{1}{\mathrm{BS}} \sum_{x \in B} \partial L(x,y; \mathbf{w}^{t}) / \partial \mathbf{w}$$

Time to perform distributed DNN training

Image Courtesy: https://www.freepik.com/premium-vector/cartoon-thinking-man-with-question-mark-think-bubble_5494827.htm

Distributed Training Strategies

- Data Parallelism
  - Centralized and Decentralized
  - Synchronous and Asynchronous
- Model Parallelism

DNN Training through Data Parallelism

- Most common
- Divide input data among workers running on Computing Units (CUs)
- Process multiple data partitions in parallel
- Replicate DNN model on each CU
- Computing Units
  - Single or multiple CPUs
  - Single GPU or hardware accelerator (e.g., FPGA, TPU)
  - Multiple GPUs on single node
  - Multiple GPUs on cluster of nodes; each node with multiple GPUs

DNN Training through Data Parallelism

- **Epoch**: One pass of the full data through DNN
- **Mini-batch**: Fixed number of training samples from input data (64, 128…)
  - Number of Mini-batches – #Training Samples (N) / Mini-batch size (B)

For epochs 1 to N_Epochs do

Shuffle the input data

For each epoch do

For each mini-batch do (N/B batches)

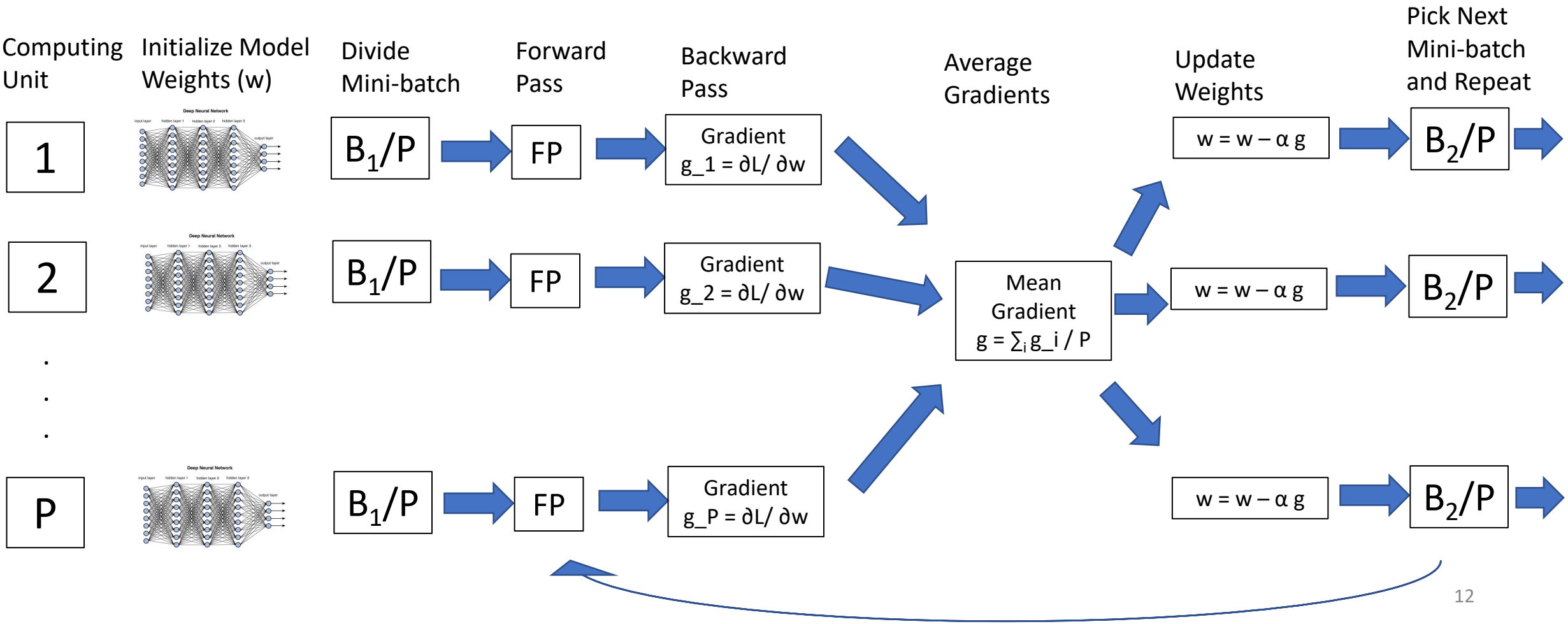
1. Feed it to DNN (forward pass) and Compute Loss
2. Calculate the mean gradient of the mini-batch (backward pass)
3. Use the mean gradient to update the weights (Optimization)

Data Parallelism

For each mini-batch do (N/B batches)

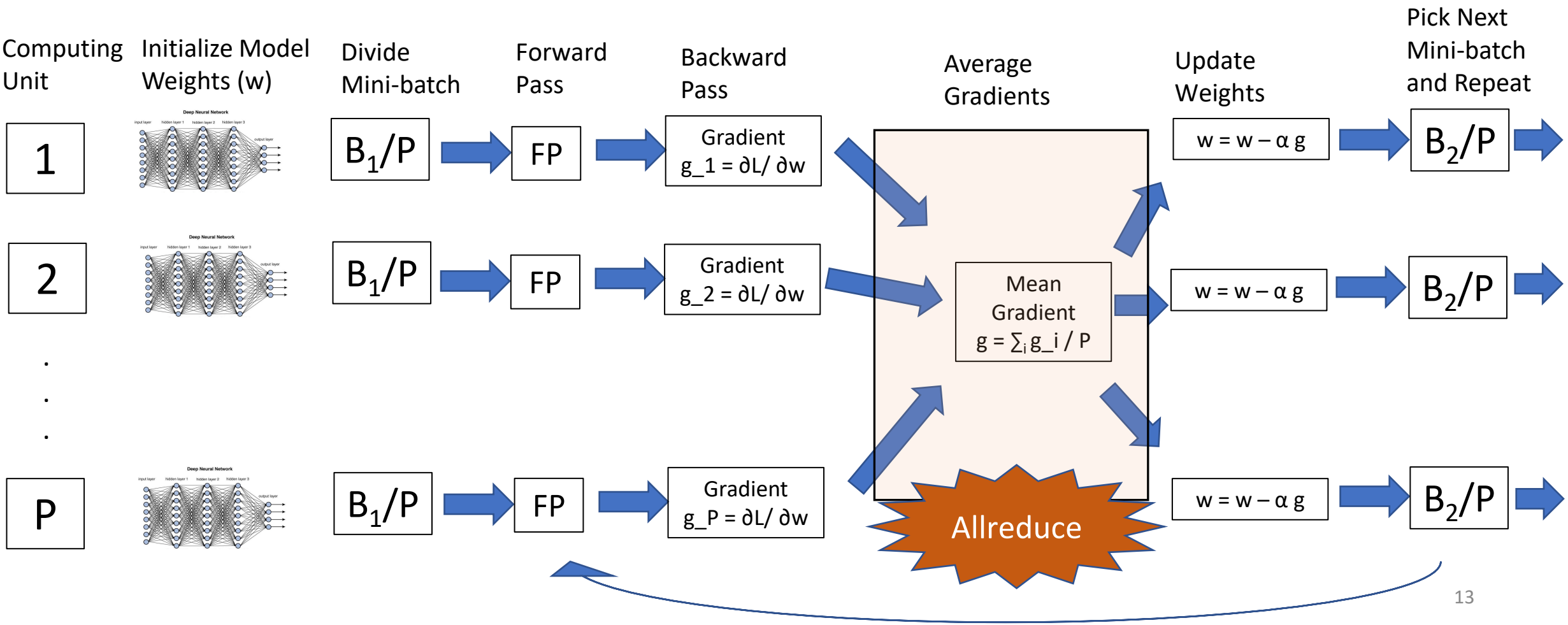
1. Feed it to DNN (forward pass) and Compute Loss
2. Calculate the mean gradient of the mini-batch (backward pass)
3. Use the mean gradient to update the weights (Optimization)

- Divide data among Computing Units (CUs)
- Replicate model on each CU

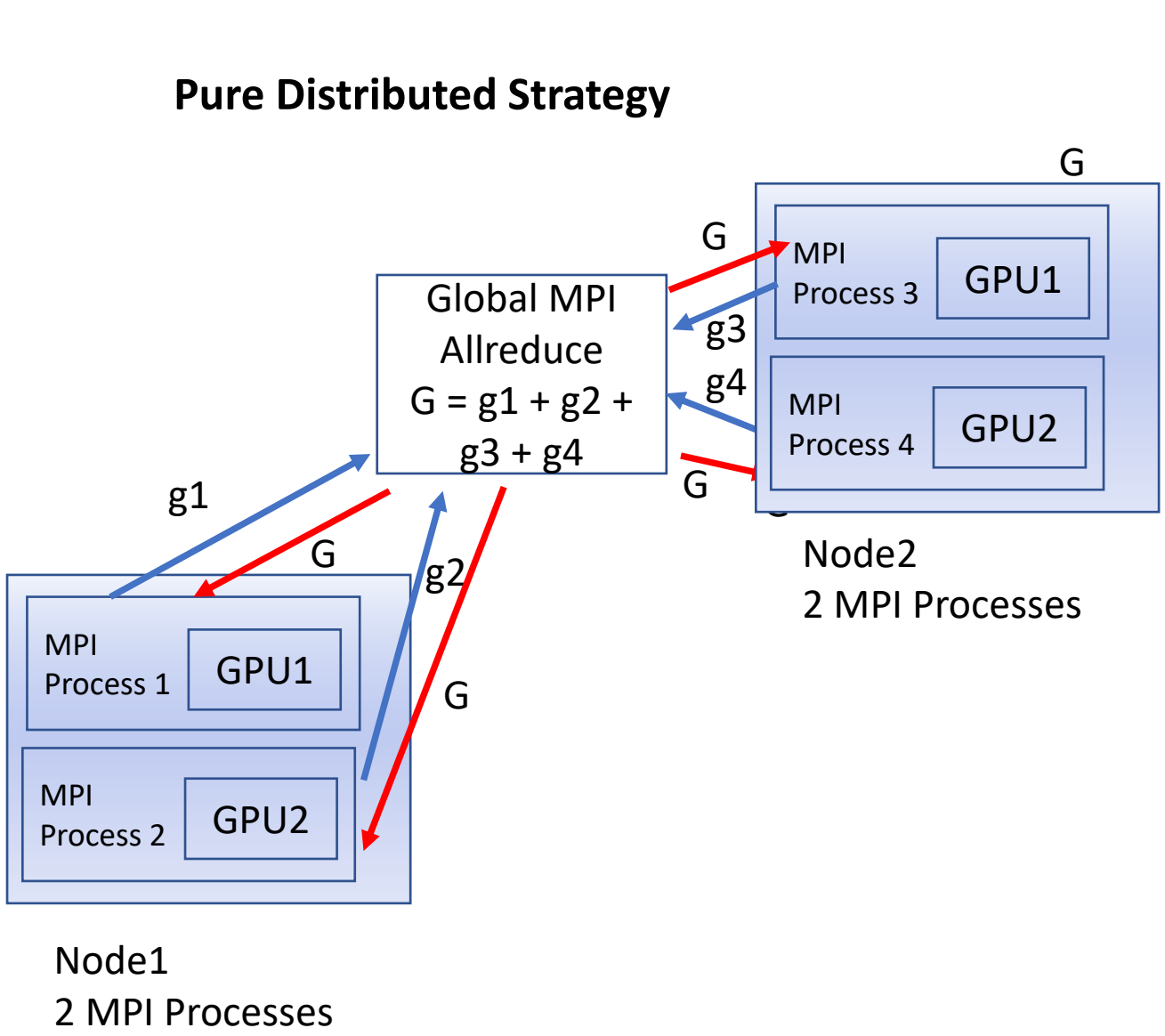


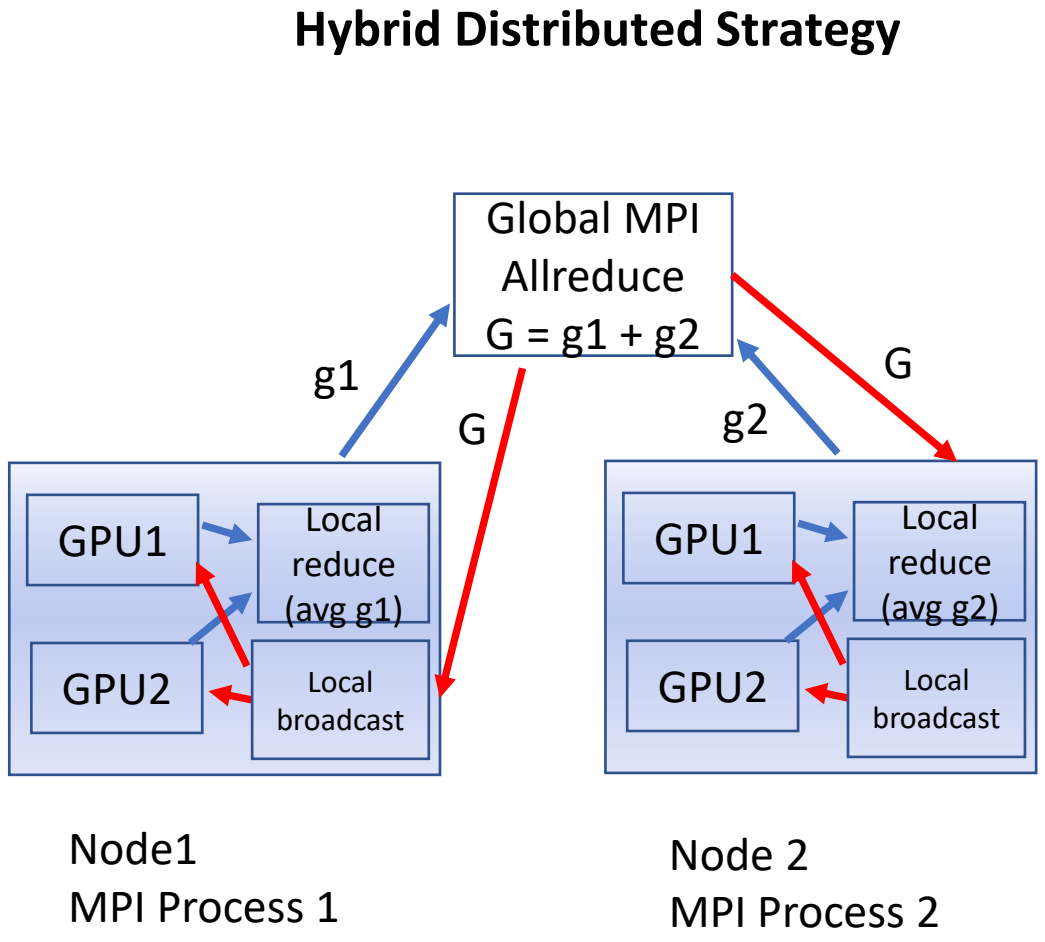
Data Parallelism

Decentralized form



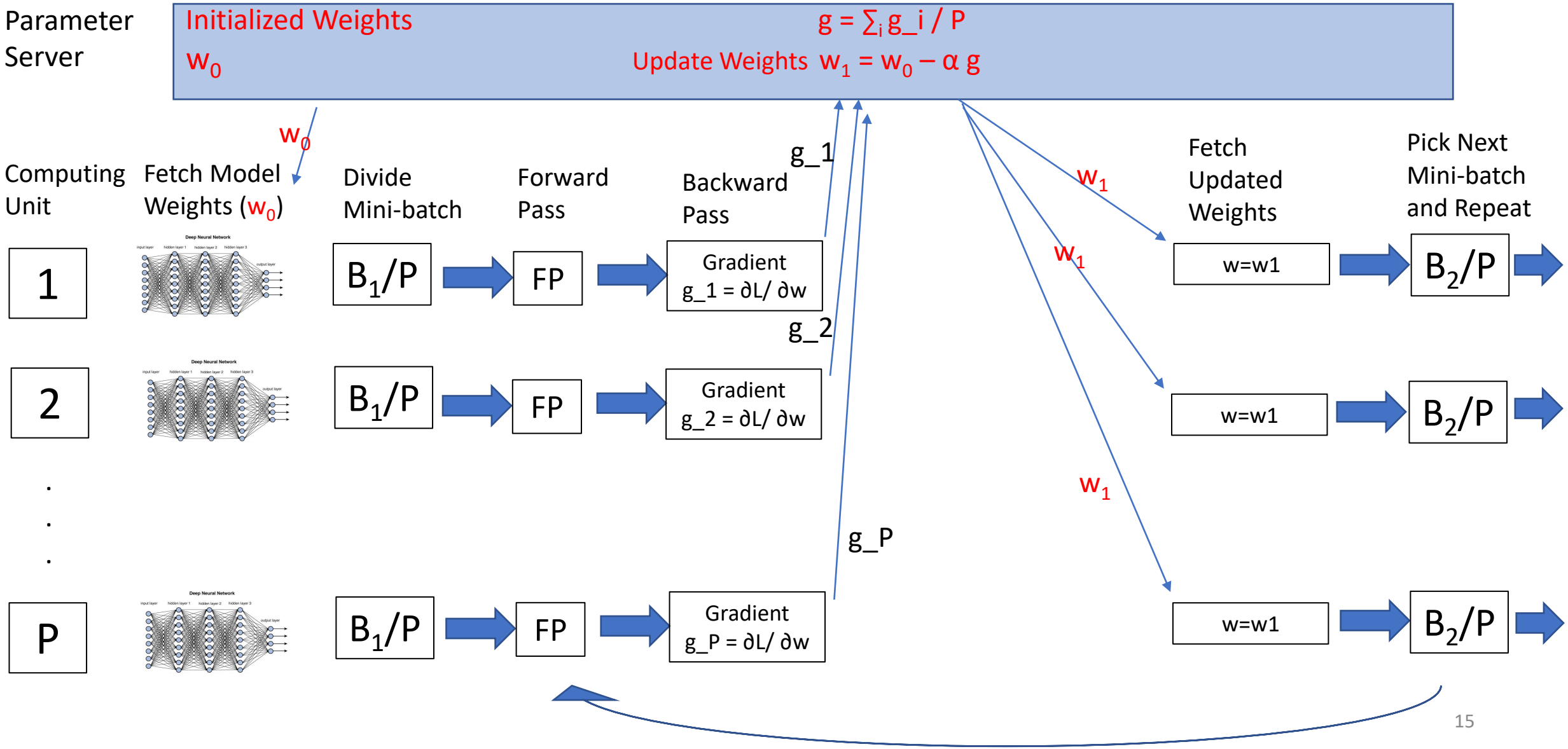
Data Parallelism on cluster of GPU nodes (with MPI)



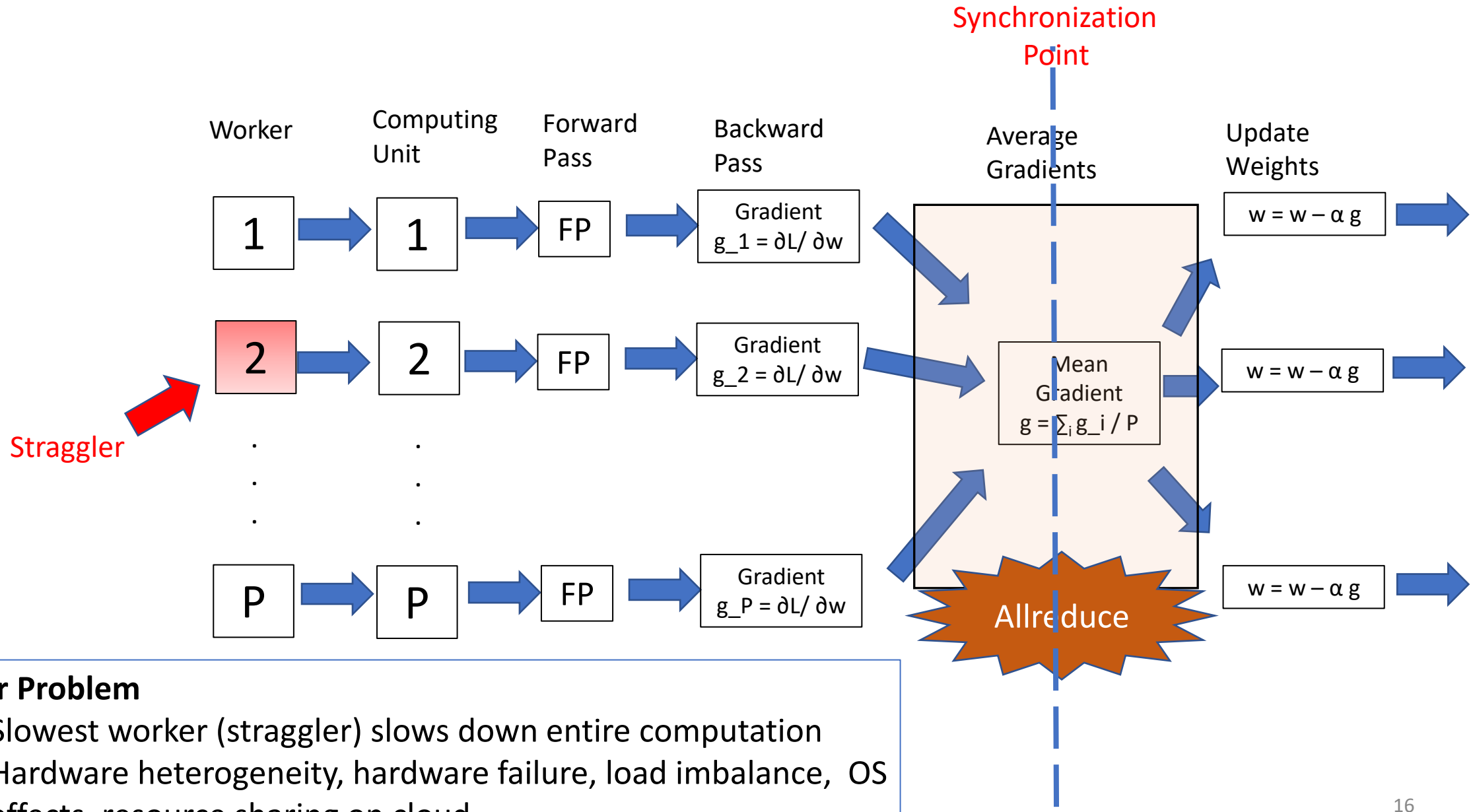


Data Parallelism

- Centralized form



Synchronous Data Parallelism

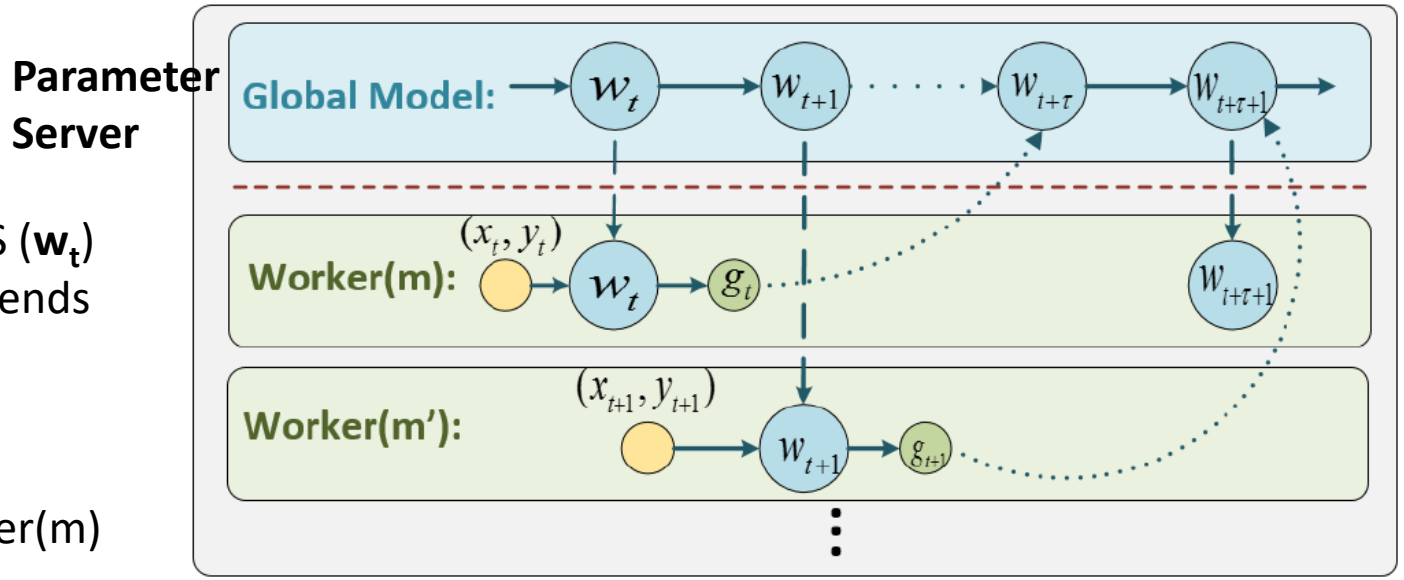


Straggler Problem

- Slowest worker (straggler) slows down entire computation
- Hardware heterogeneity, hardware failure, load imbalance, OS effects, resource sharing on cloud

Asynchronous Data Parallelism

- Model weights updated asynchronously without waiting for all workers
- Achieved through Parameter Server (PS)
- Model replicated on workers, not identical
- Stale gradient problem
- Aka Async-SGD or ASGD



- Parameter server stores global model (**w**)
- Worker(m) gets a snapshot of global model from PS ($\mathbf{w_t}$)
- Worker(m) computes it local gradient (g ($\mathbf{w_t}$)) and sends to PS
- PS updates the global model without waiting

$$\mathbf{w}_{t+\tau+1} = \mathbf{w}_{t+\tau} - \alpha\, g(\mathbf{w}_t)$$

- Global model updated using **stale gradient** of worker(m)

Sync v/s Async Data Parallelism: Training time and Error

- Sync
  - More time to converge due to stragglers
  - Reaches higher accuracy
- Async
  - Faster convergence
  - Reaches lower accuracy

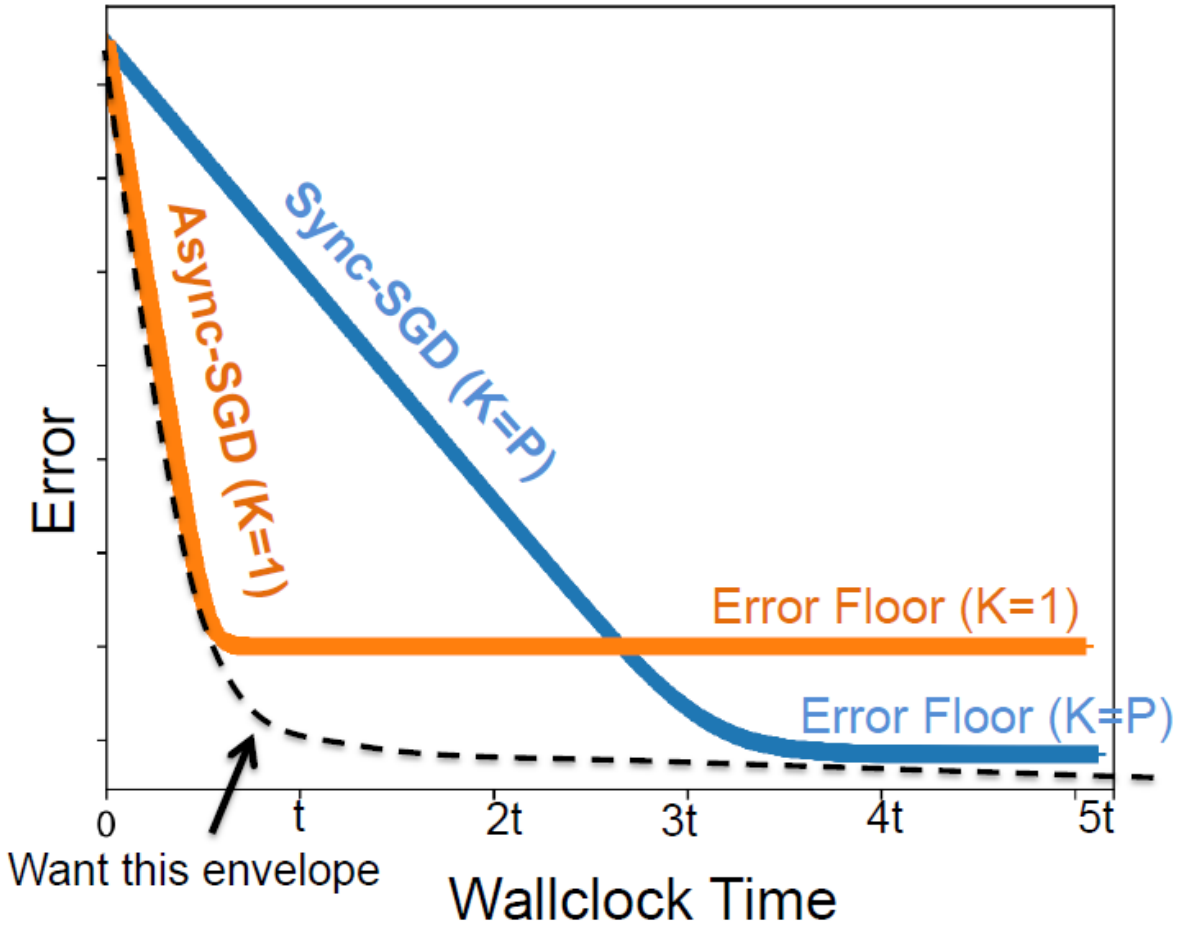
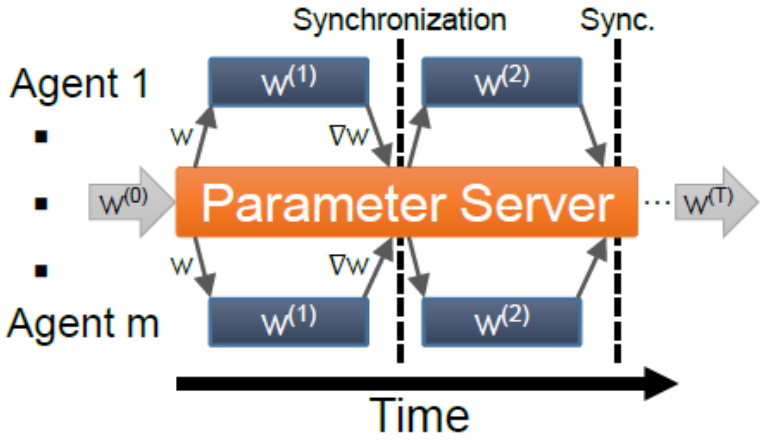
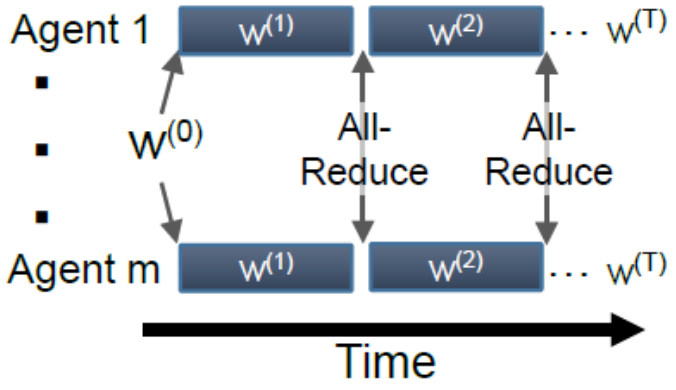


Figure Courtesy: Slow and Stale Gradients Can Win the Race, https://arxiv.org/abs/2003.10579

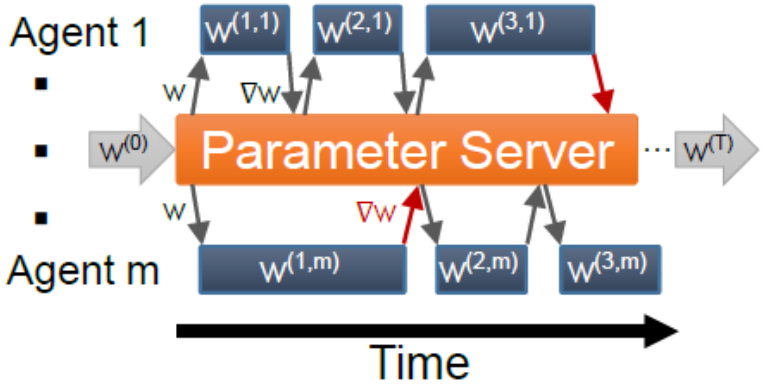
Data Parallelism Strategies (Model consistency, Centralization)



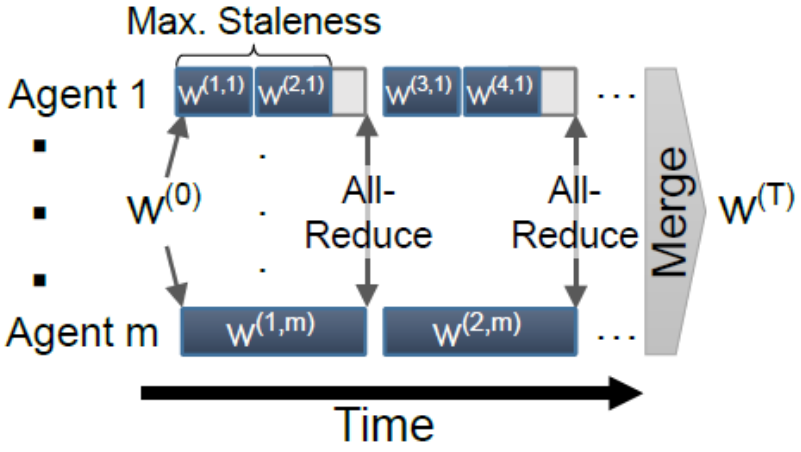
(a) Synchronous, Parameter Server



(b) Synchronous, Decentralized



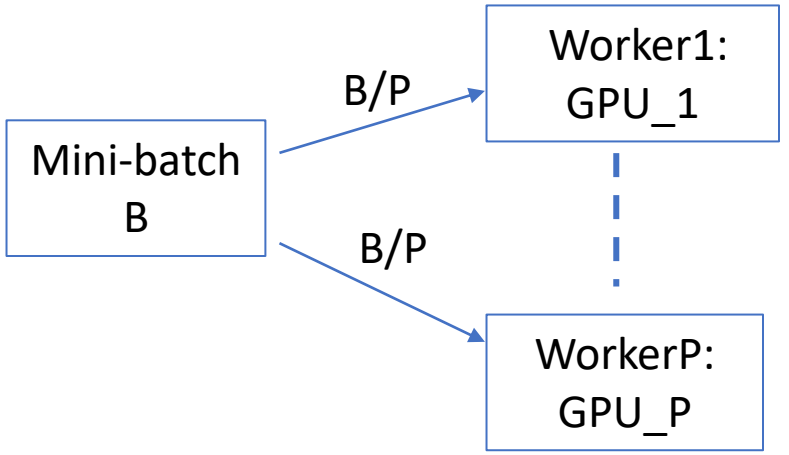
(c) Asynchronous, Parameter Server



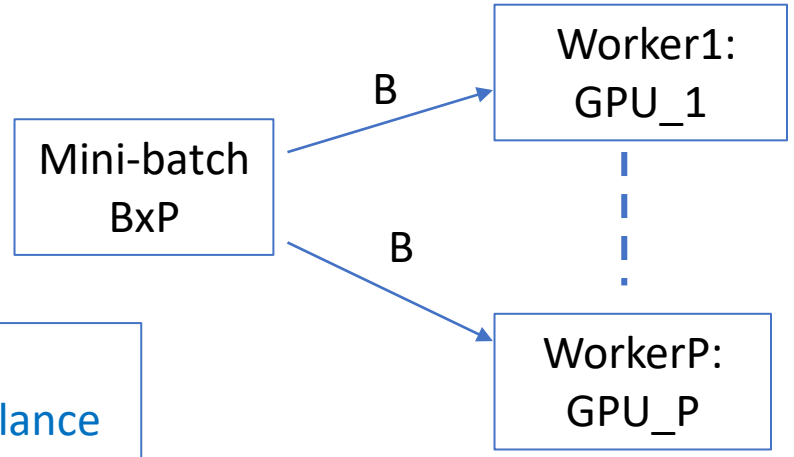
(d) Stale-Synchronous, Decentralized

Data Parallelism and Mini-batch Size

- Fixed total mini-batch size (B)
  - Partition across workers
  - B/P mini-BS per worker



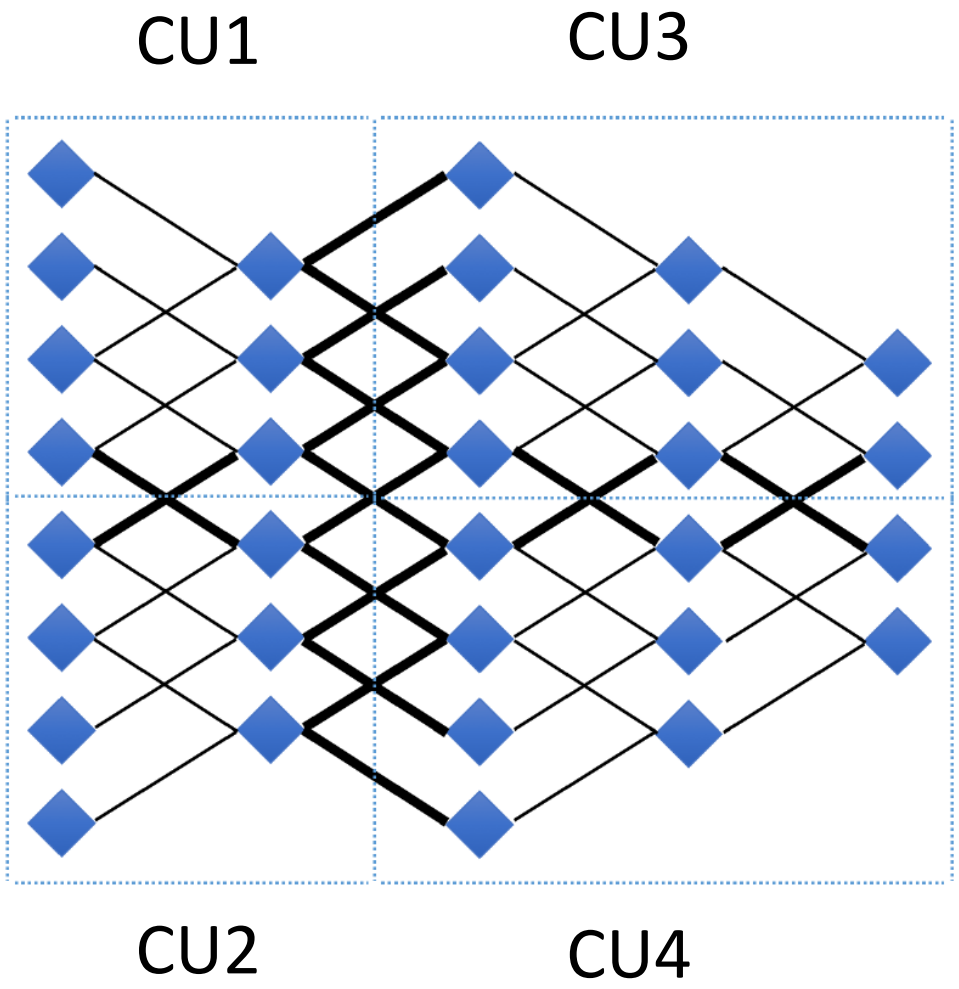
- Fixed mini-batch size per worker (B)
  - Total mini-batch size = B x P
  - Total mini-BS scales with P



- GPU Utilization
- Compute v/s Communication overhead balance
- Input data size (# training samples)
- Model accuracy

Model Parallelism

- Partition DNN model among Computing Units (CUs)
- Replicate mini-batch on CUs
- Communication with neighboring CUs sharing connections (in bold)



5-layer DNN partitioned into 4 CUs

Image Courtesy: https://arxiv.org/abs/1709.05011

Part 2:
Large Scale DNN Training Example:
Training Imagenet-1K with Resnet-50

Imagenet-1K Classification Task: Background

- Dataset: Imagenet-1K
  - Dataset of images labeled into 1000 classes
  - ~1.28M Training Images and 50K Validation images
- DNN Model : Resnet-50
  - 50-layer Convolutional Neural Network (CNN)
  - ~25M Parameters (~100MB gradient size)
- Evaluation metric: Classification Accuracy on validation set
  - Accuracy = Fraction of images correctly classified (= 1 – classification error)

State of the art (Till Nov 2018)

Time to train Resnet-50 over Imagenet-1k dataset with 90 epochs

Table 1 : Training time and top-1 1-crop validation accuracy with ImageNet/ResNet-50

	Batch Size	Processor	DL Library	Time	Accuracy
He et al. **[7]**	256	Tesla P100 x8	Caffe	29 hours	75.3%
Goyal et al. **[1]**	8K	Tesla P100 x256	Caffe2	1 hour	76.3%
Smith et al. **[4]**	8K→16K	full TPU Pod	TensorFlow	30 mins	76.1%
Akiba et al. **[5]**	32K	Tesla P100 x1024	Chainer	15 mins	74.9%
Jia et al. **[6]**	64K	Tesla P40 x2048	TensorFlow	6.6 mins	75.8%
Sony[2]	**34K→68K**	**Tesla V100 x2176**	**NNL**	**224 secs**	**75.03%**

Reference - [1] **Training Imagenet in 1 hour (https://arxiv.org/abs/1706.02677)**
[2] ImageNet/ResNet-50 Training in 224 Seconds (http://arxiv.org/pdf/1811.05233v1)

- Parallelization Strategy - **Data Parallel Synchronous SGD**
- Main Challenges:
  - **Accuracy drop** when training with very large mini-batch sizes
  - **Communication overhead** of gradient aggregation over large number of GPUs

Handling impact of mini-batch size on accuracy

- Large mini-batch size causes optimization difficulty
- Learning Rate (LR) adjustment
  - Linear scaling rule

  LR(P*B) = P * LR(B)
  - Gradual warmup
  - Limit on max BS
- Don’t decay LR, increase BS
  - LR/f equivalent to f x B

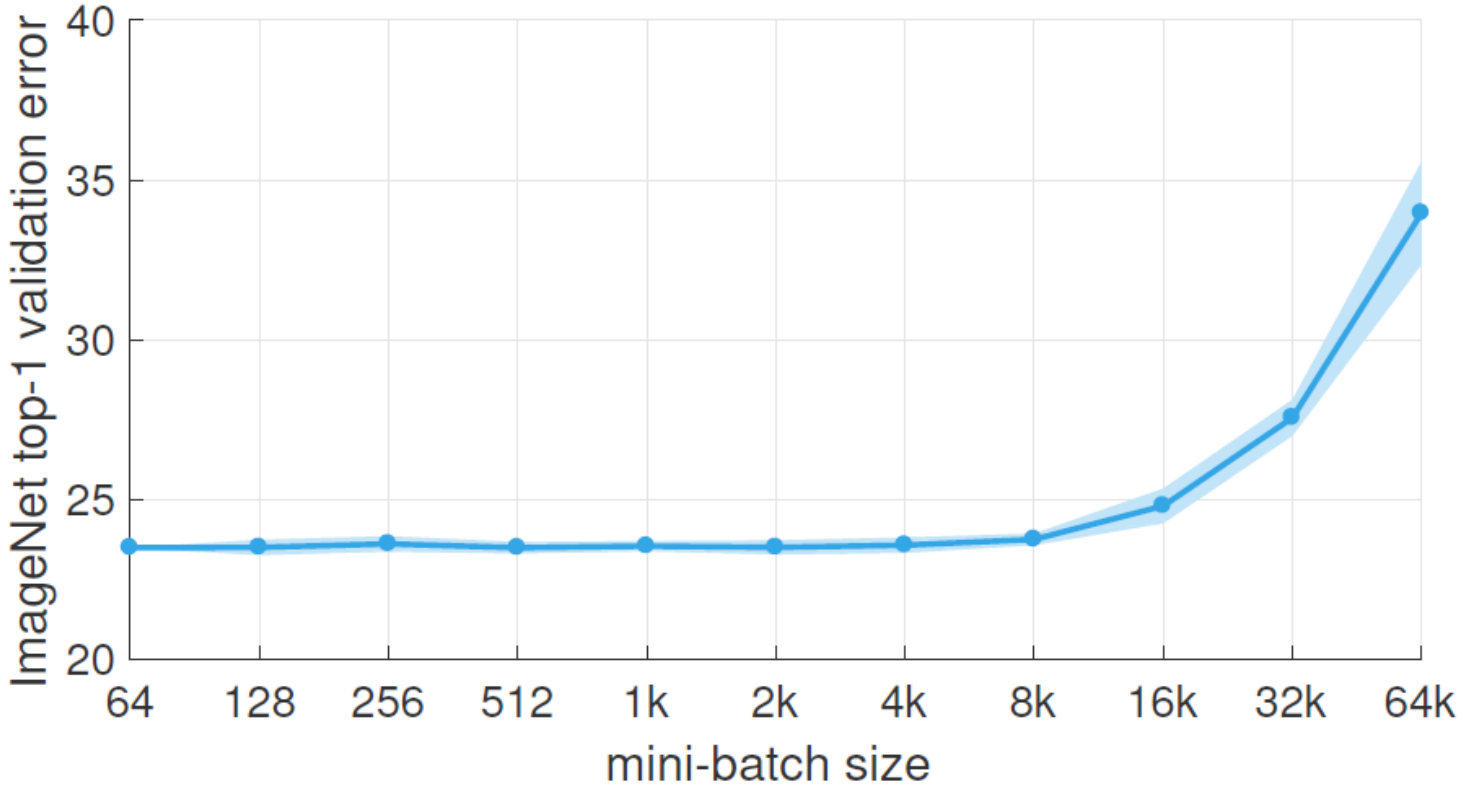
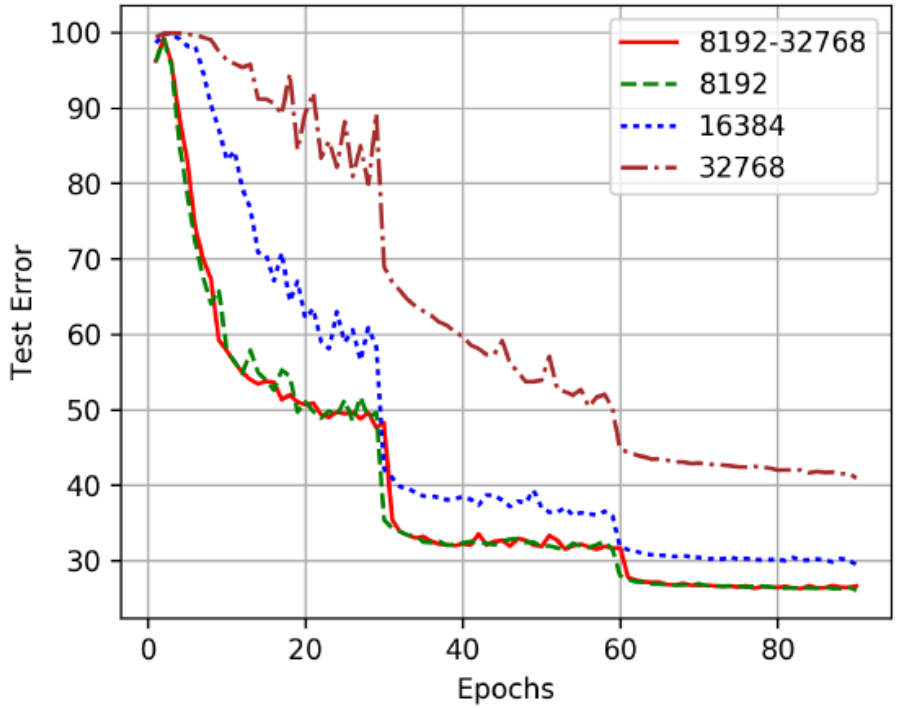


Figure 1. **ImageNet top-1 validation error *vs.* minibatch size.**

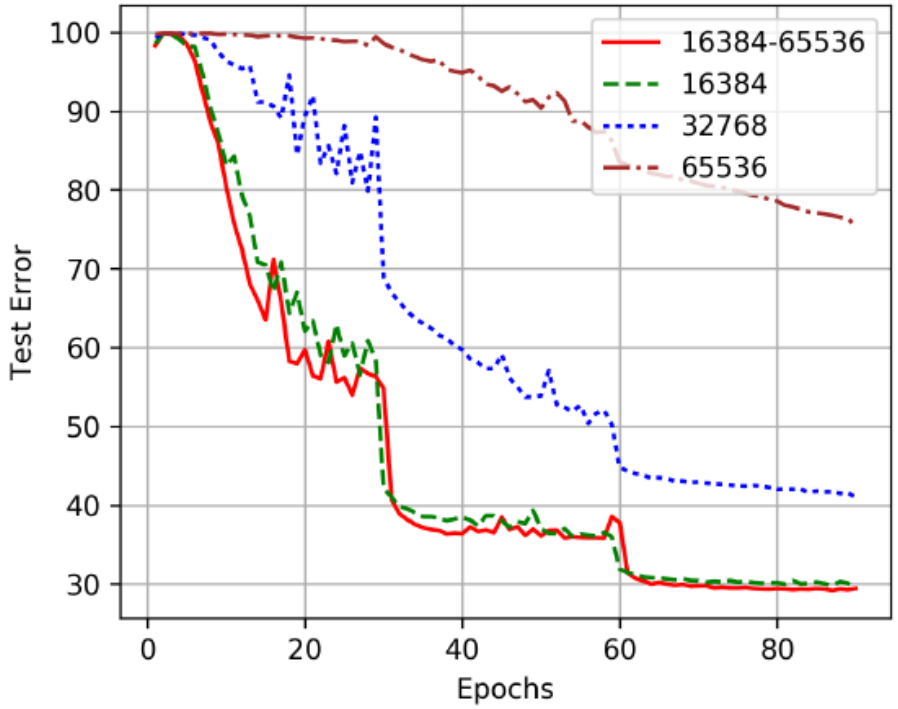
Fig Source : Training Imagenet in 1 hour (https://arxiv.org/abs/1706.02677)

Reference : Don’t decay LR, increase BS, https://arxiv.org/pdf/1711.00489v1.pdf

Handling impact of mini-batch size on accuracy



(a) ResNet-50, starting batch size 8192.



(b) ResNet-50, starting batch size 16384.

Figure 6: Comparison of ImageNet test errors curves for adaptive versus fixed batch sizes with LR warmup.

Fixed Batch size – LR decayed by factor of 0.1 every 30 epochs
Adaptive Batch size – LR decayed by factor of 0.2, BS doubled every 30 epochs

Handling Communication overhead of Gradient Aggregation (allreduce)

- Faster Interconnect
- Custom Allreduce
  - Alternatives to MPI - gloo, nccl
- Overlapped communication (allreduce) with backpropagation
- Gradient Quantization
  - Use lower precision gradients (16-bit, 8-bit, etc.)
- Gradient Sparsification
  - Only communicate gradients larger than a threshold

Take Away

Part 1 – Distributed DNN Training

- DNN and DNN Training Recap
- DNN Training Parallelization Strategies
  - Data Parallel
    - Synchronous, Asynchronous
    - Centralized, Decentralized
  - Model Parallel

Part 2 - Large Scale DNN Training Example

- Training Imagenet-1K with Resnet-50
- Challenges and solutions
  - Accuracy drop
  - Communication overhead