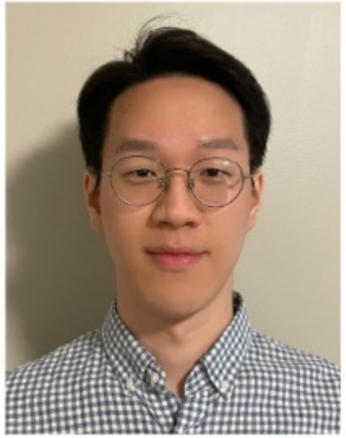
Efficient Strong Scaling Through Burst Parallel DNN Training

MIT CSAIL


Seo Jin Park


Josh Fried


Sunghyun Kim

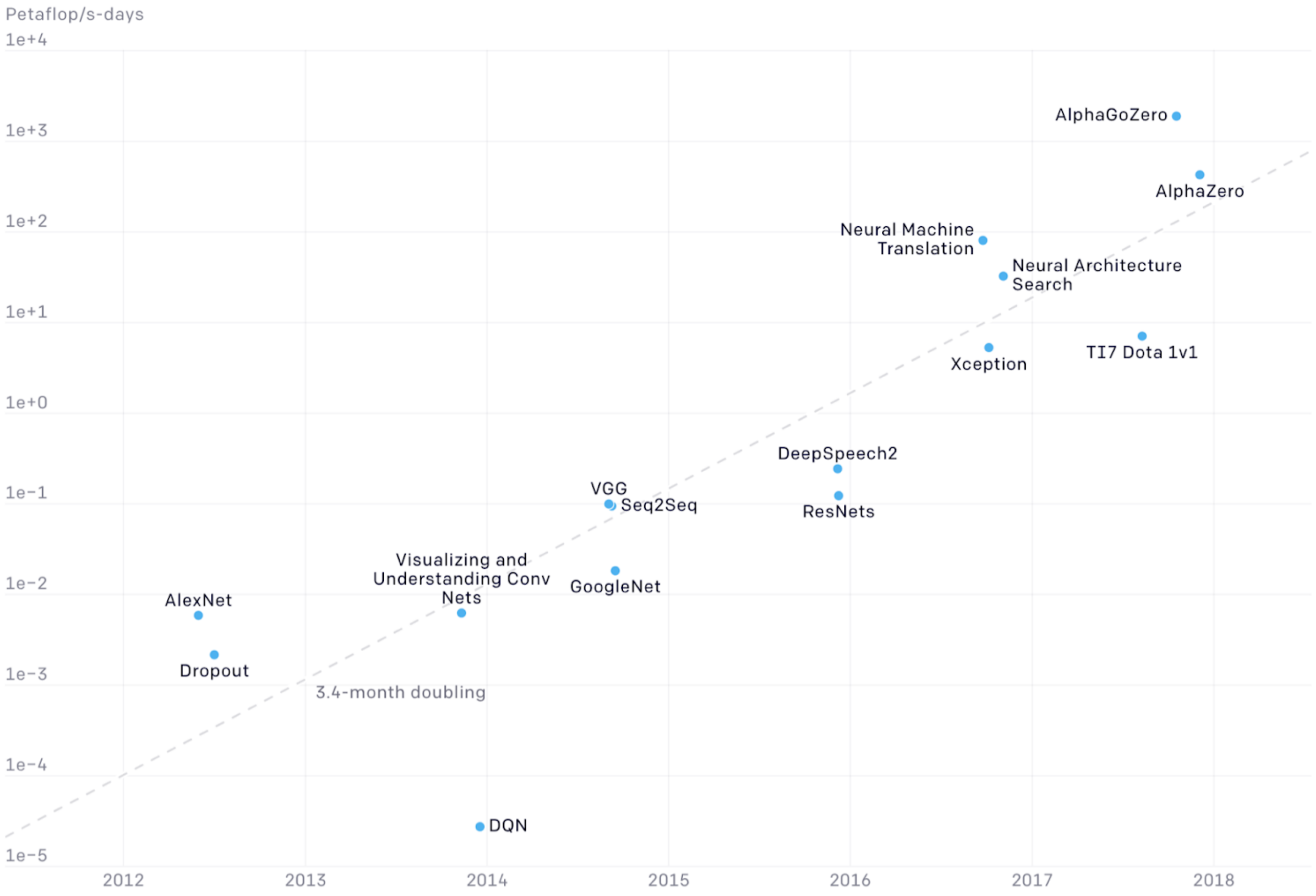

Mohammad Alizadeh


Adam Belay

August 30, 2022
MLSys 2022

Trend: DNN model complexity is increasing over time

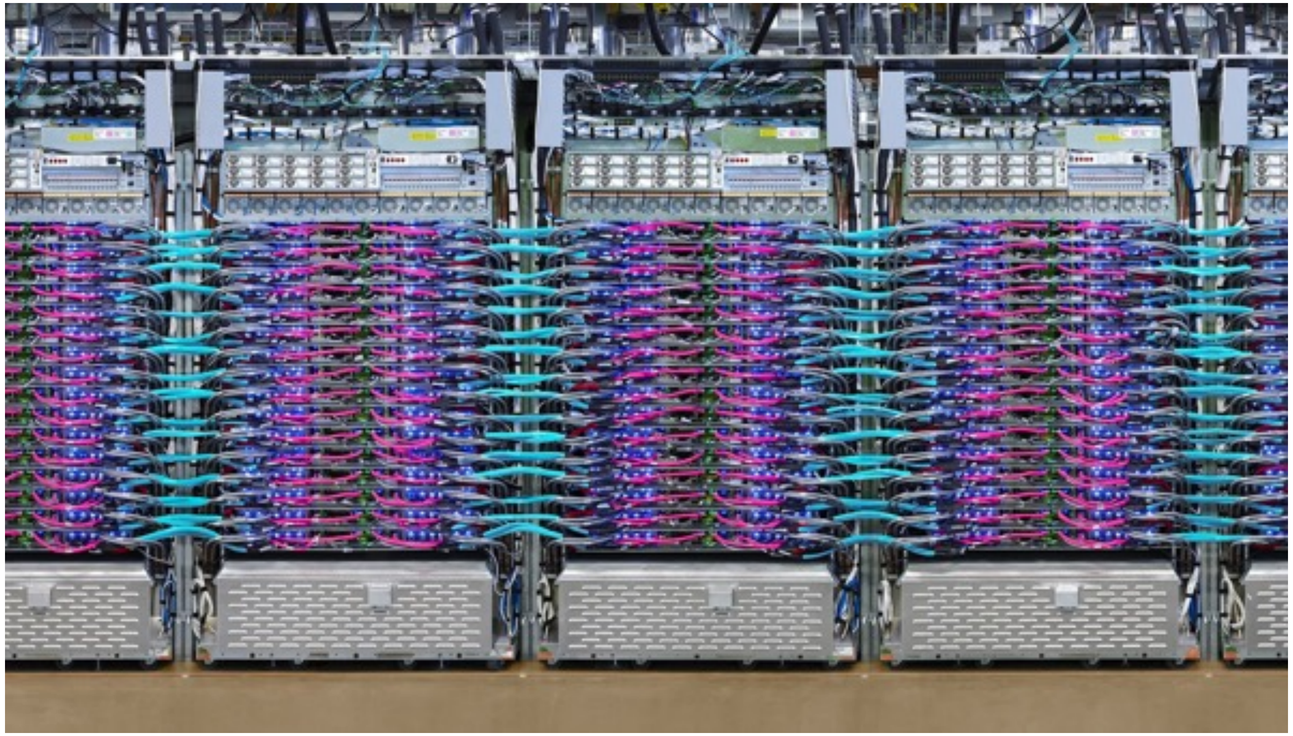
Training cost of state-of-the-art DNN models over time



- **300,000x** increase in compute over 5 years
- Improvements in GPUs (and TPUs) have only partially closed this gap

Requirement: Increasingly large training clusters

- Example: Facebook is using 256+ GPUs to train its DNN models
- Example: Google offers a 512-core pod slice (~2 Million $ per year)
- Hard problem: How to scale DNN training?

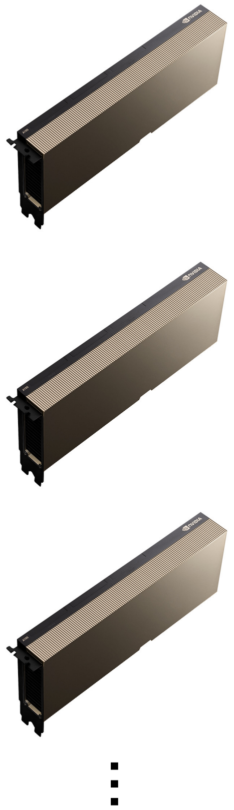


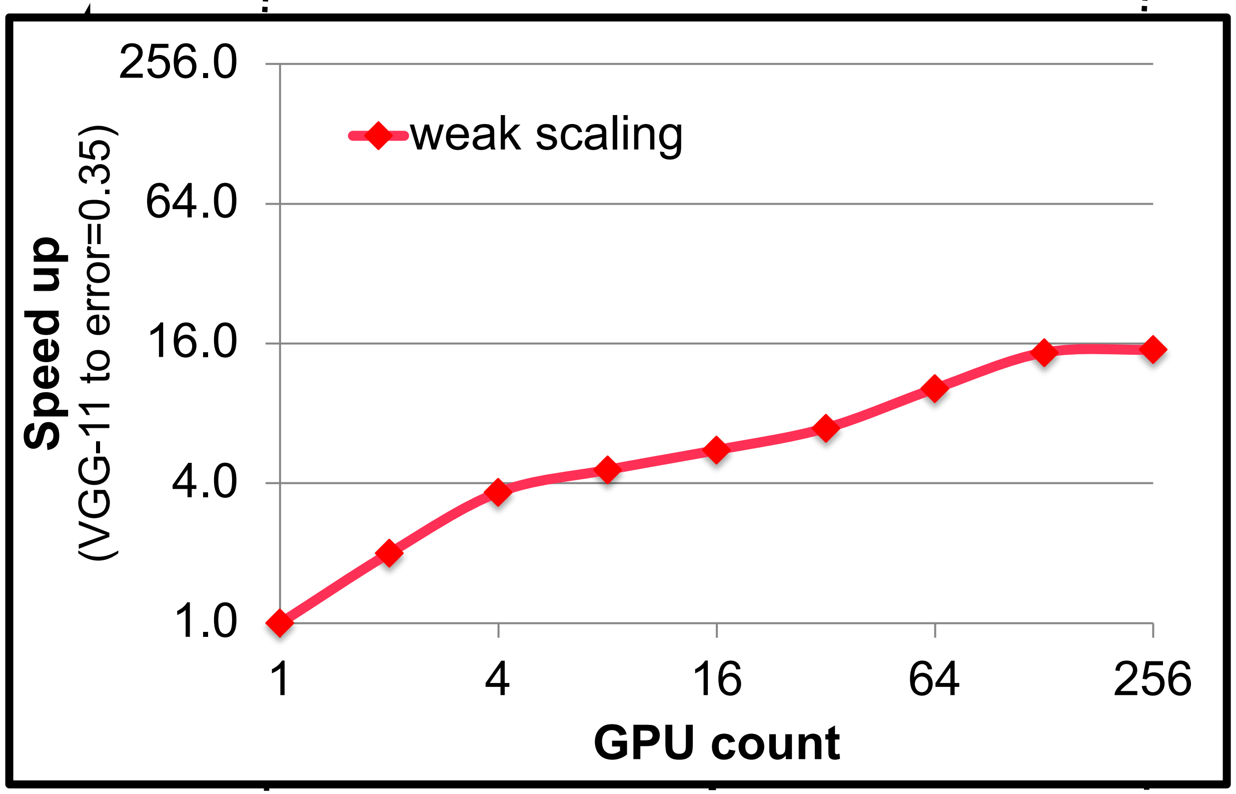
Google’s Cloud TPU v3 Pod

Source: https://cloud.google.com/tpu

Conventional approach: Scale batch size with cluster

- **Weak scaling** (data parallelism + larger batches)





The batch size grows larger & larger

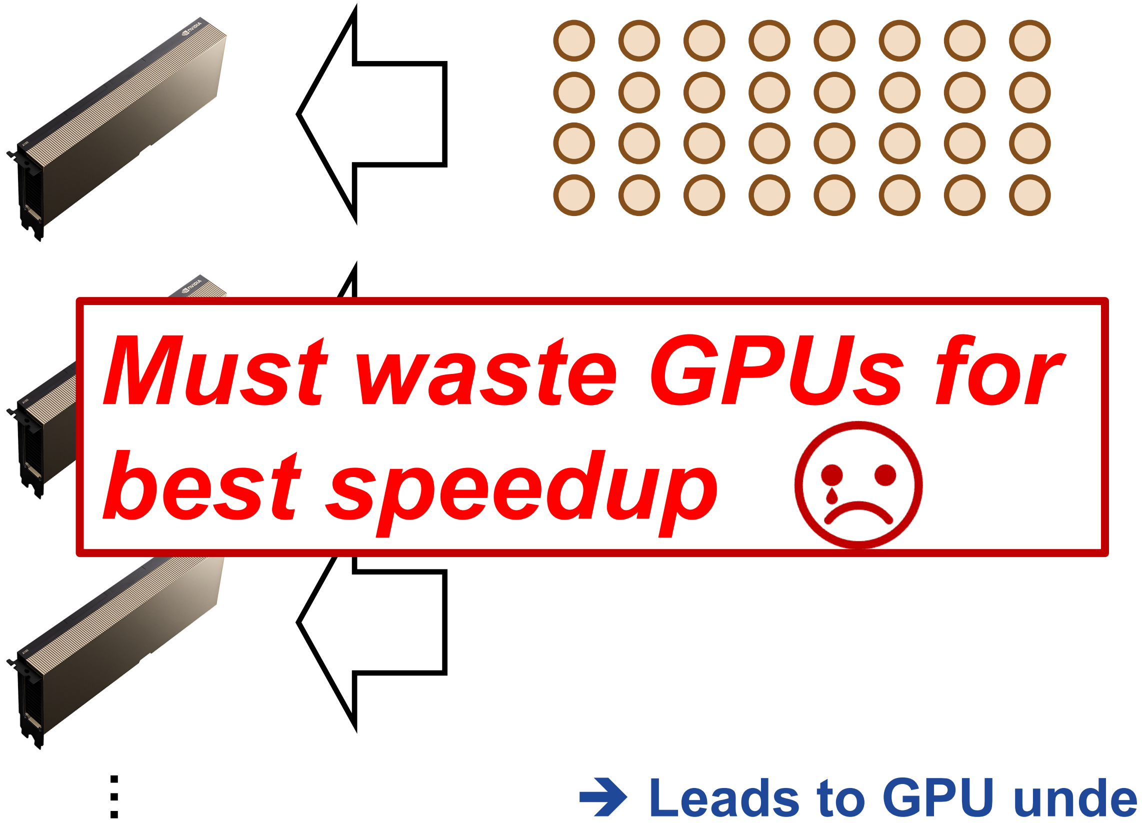
Some benefits

- Increases throughput (samples / sec) & keeps utilization high
- Amortizes communication cost

<u>BUT</u> too large batches reduce the statistical efficiency of samples

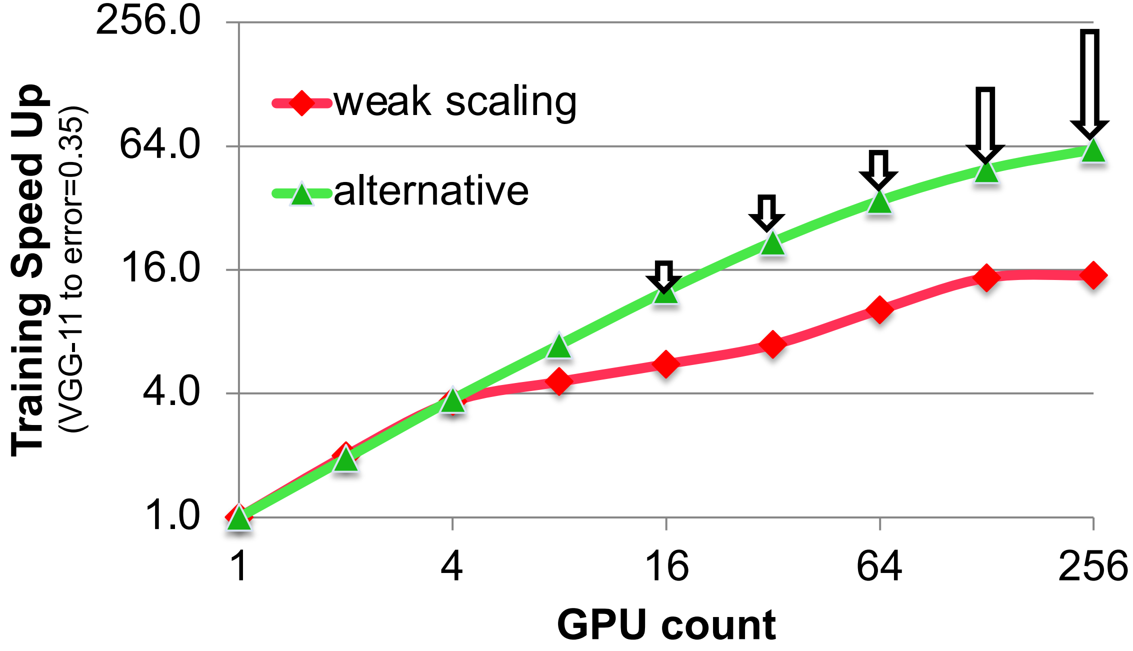
Alternative: scale by reducing # samples per GPU

- **Strong scaling** (distribute samples to many GPUs)



Must waste GPUs for best speedup

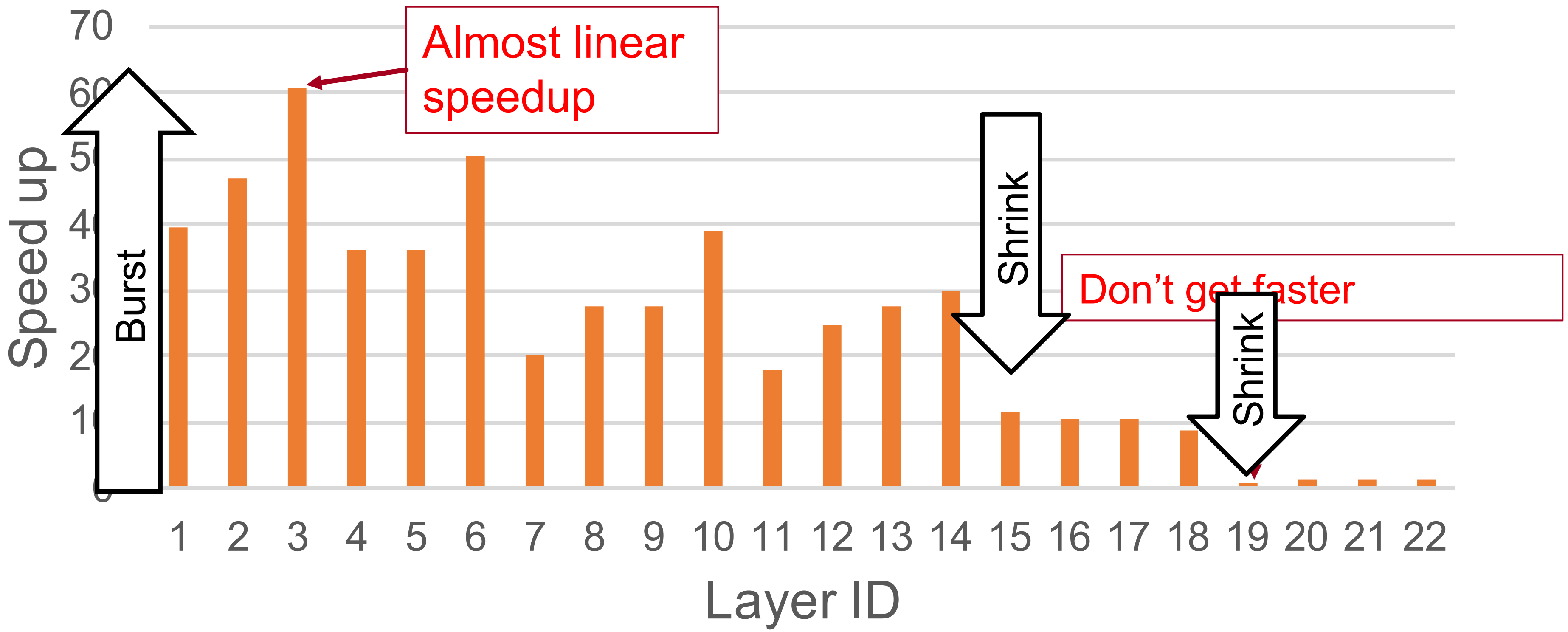
- **Speed up by reducing iteration time**



➔ **Leads to GPU underutilization**
(Must sacrifice efficiency for the best speedup)

Opportunity 1: Unevenness in Scalability

- **GPUs are wasted for unscalable layers**



Scalability of layers in VGG16 when scaled with 64 GPUs
(128 samples/ iteration → 2 samples / iteration)

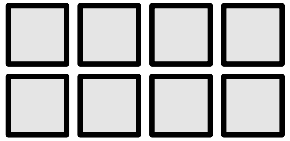
Opportunity 2: Existence of Small Jobs

- **Large GPU cluster has many users and tasks**
- **Two kinds of training tasks exist in large GPU cluster**

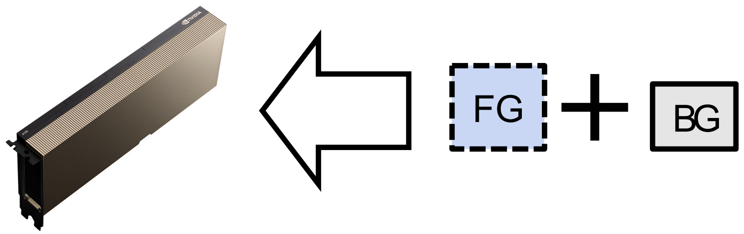
Large scale jobs

- Ex) Training big model with large dataset
- Must scale to 100+ GPUs
- Speed is important (foreground only)

Small scale jobs



- Ex) Quick test with small dataset
- Fits in < 1 GPU
- Can tolerate slower training (May run on background)

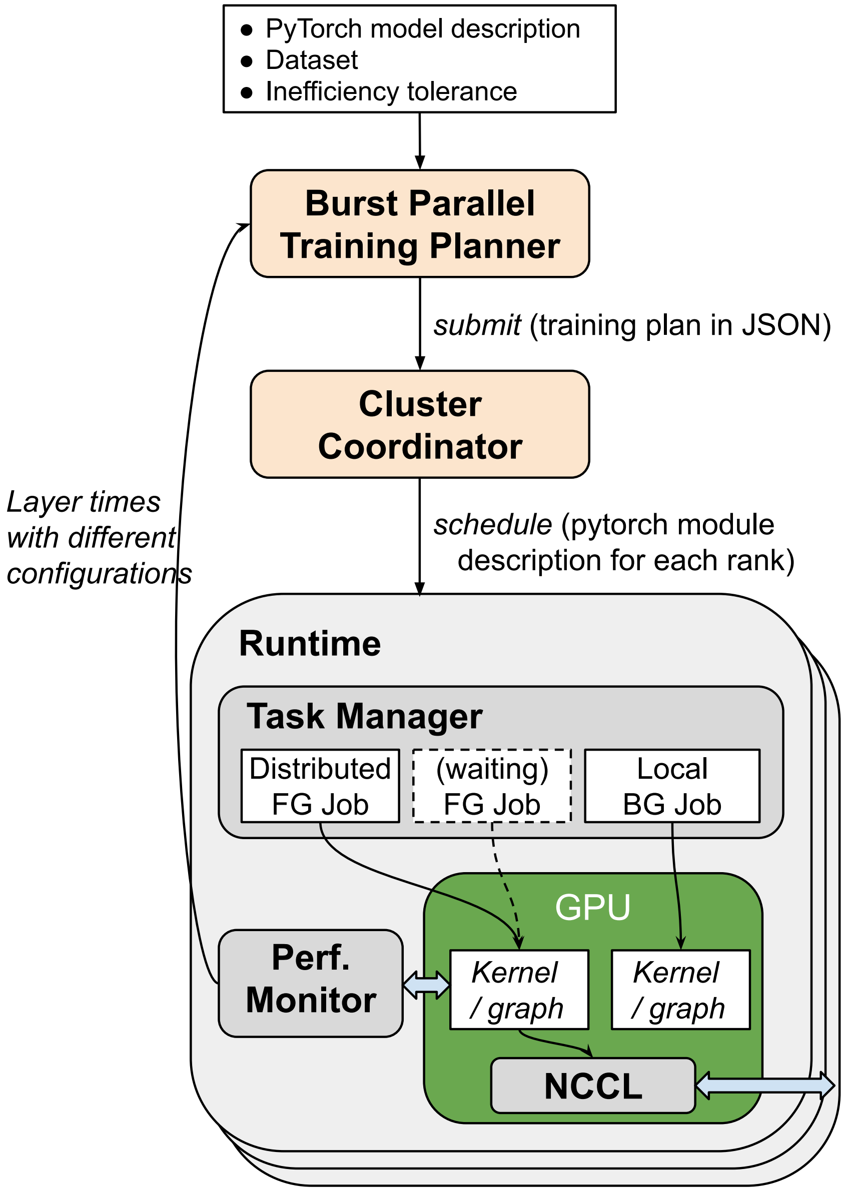


Our Proposed Solution

Enable **high speedup** while achieving **high efficiency** for the entire cluster:

1. **Burst parallelism**
   - Map each layer to an optimal set of GPUs
2. **GPU multiplexing**
   - Run a background job while GPU is idle for foreground

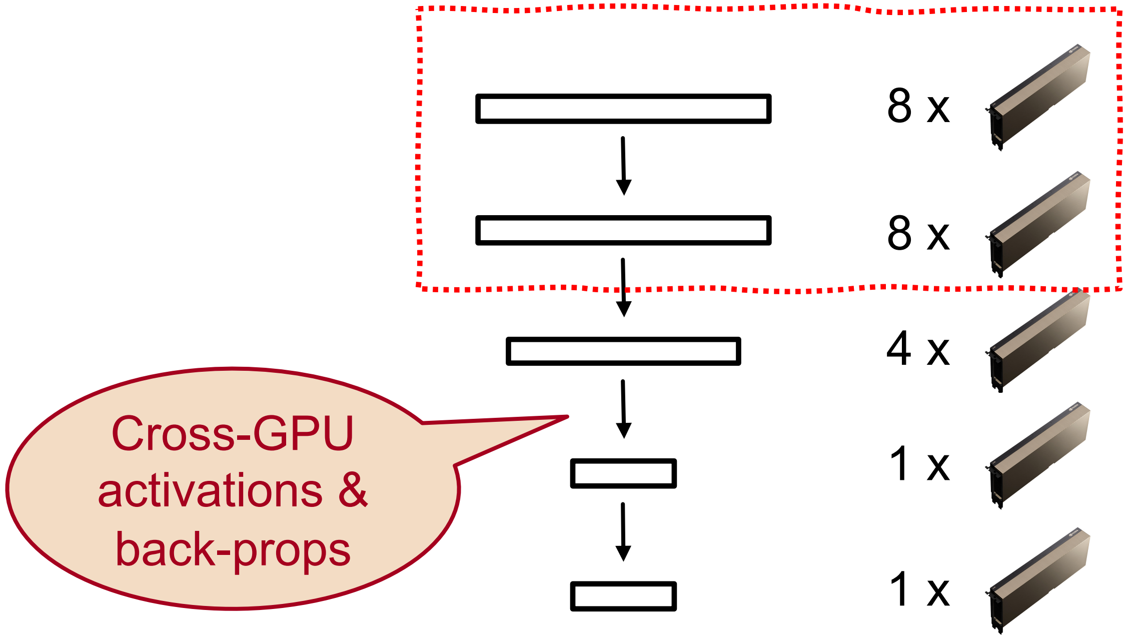
DeepPool system overview



- **Input:** PyTorch-like model implementation, dataset, and *inefficiency tolerance*
- **Burst parallel training planner**
  - Decides the scaling of each layer to **stay efficient**
  - Profiles each layer with different batch sizes
    (Planner also supports SOAP model parallelism)
- **Runtime** (for each GPU)
  - Manages & schedules jobs to GPU
    - 1 distributed FG task, 1 local BG task
  - Uses C++ frontend of PyTorch & NCCL

Burst Parallel Training Planner

- **Decides the level of strong scaling of each layer**
  - Optimal global batch & available #GPUs are given by users
- **Search** by dynamic programming + graph reduction



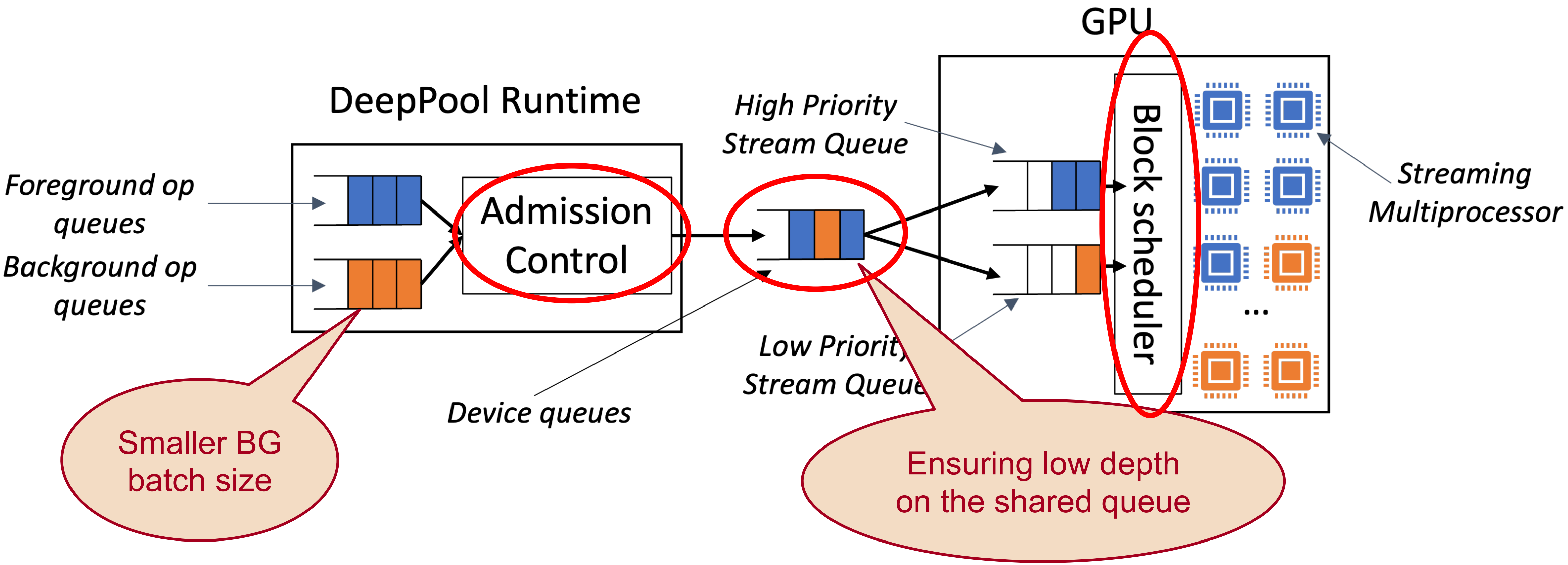
Efficiency: GPU-sec amplification

- *GPU-sec*: aggregate active GPU time / iter (like man-hour or Watt-hour)
- *GPU-sec amplification* =

$$\frac{GPU\text{−}sec\ when\ scaled}{Single\ GPU\ iteration\ time}$$

Protecting QoS while Multiplexing

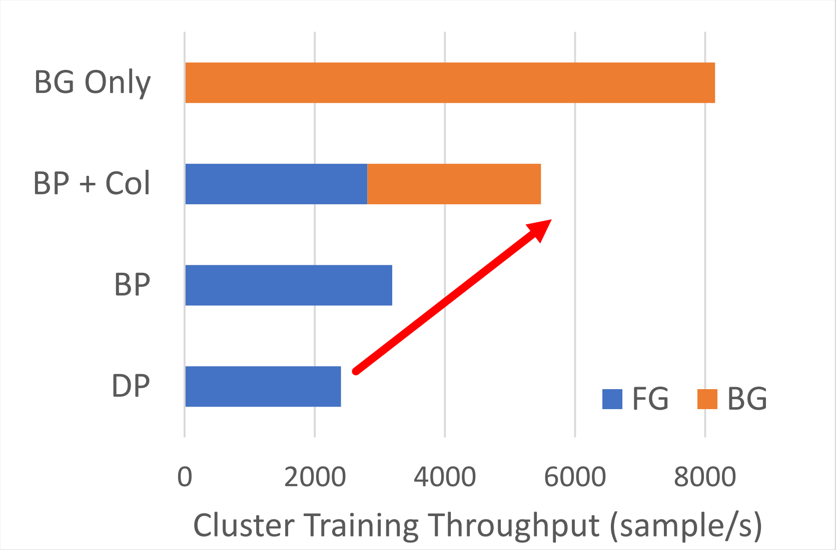
- **Used 2 NVIDIA GPU features:** CUDA streams (w/ priority), CUDA graph
- **Problem:** shared queue & non-preemptive scheduler



Evaluation

- **Workload: 3 image classification models**
  - VGG-16 (132M params), WideResNet-101-2 (127M params), Inception-V3 (24M params)
- **Hardware: DGX A100 box**
  - 8 NVIDIA A100 GPUs
  - NVSwitch (600GB/s for each GPU)
  - CUDA 11.4, cuDNN v8.2.4, NCCL 2.10.3
- **Questions**
  1. Can we improve training throughput of each GPU while strong scaling a foreground job?
  2. Does DeepPool offer better combinations of total cluster throughput and foreground speedup than statically partitioning a cluster?
  3. How do individual techniques of DeepPool enable low interference collocation?

Can we improve training throughput?



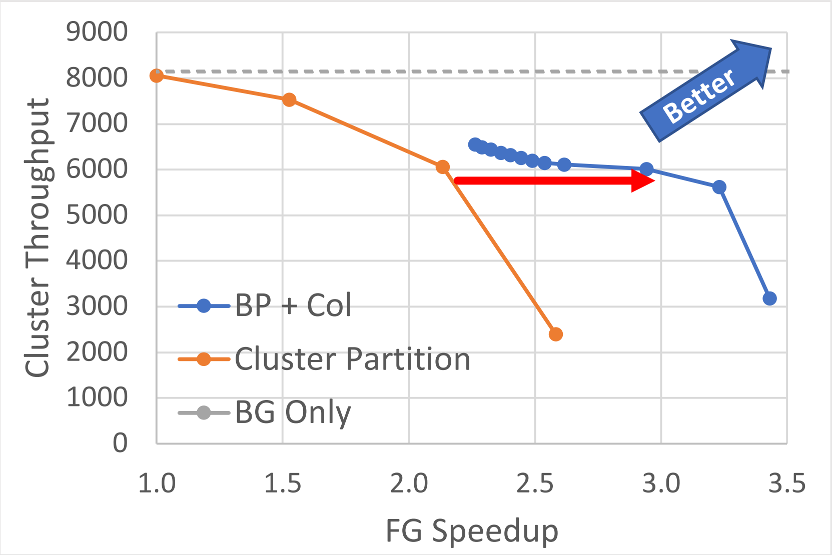
(a) VGG-16, scaling b=32

Legend

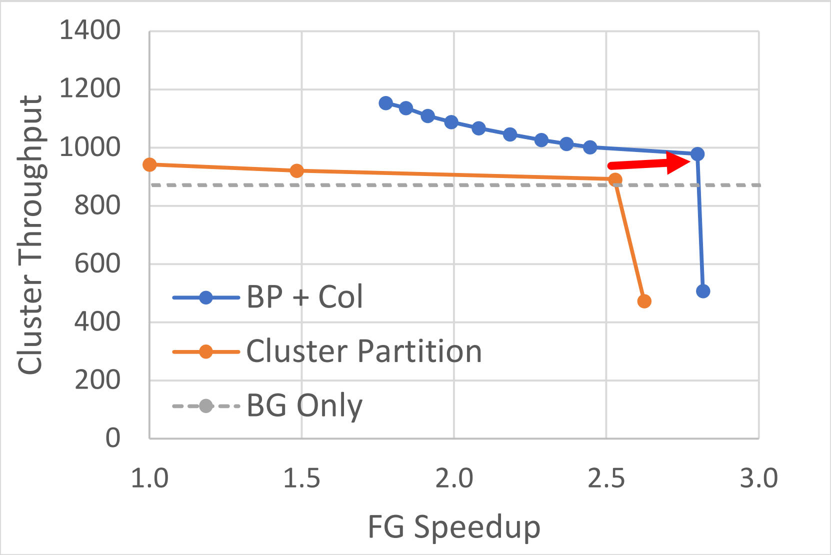
- **DP**: baseline, only data-parallel FG task by evenly splitting the global batch across 8 GPUs.
- **BP:** burst parallel training for FG task.
- **BP+Col**: collocates a low priority BG task with the burst-parallel FG job. FG and BG use the same workload.
- **BG Only**: runs the low priority BG task only (for reference)

Burst Parallelism vs. Cluster Partition

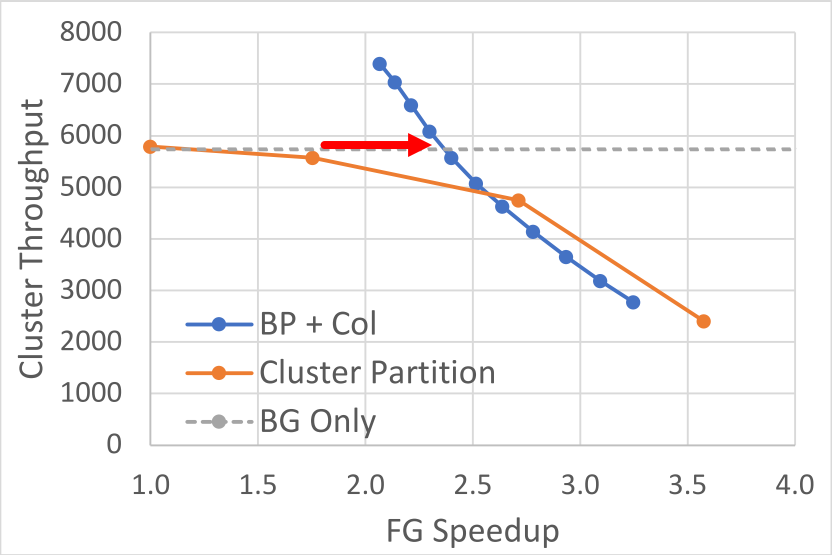
- **Baseline: partition cluster into “FG” GPUs and “BG” GPUs**
  - 4 configs: <1 FG & 7 BG>, <2 FG & 6 BG>, <4 FG & 4 BG>, <8 FG>



(a) VGG-16, scaling b=32

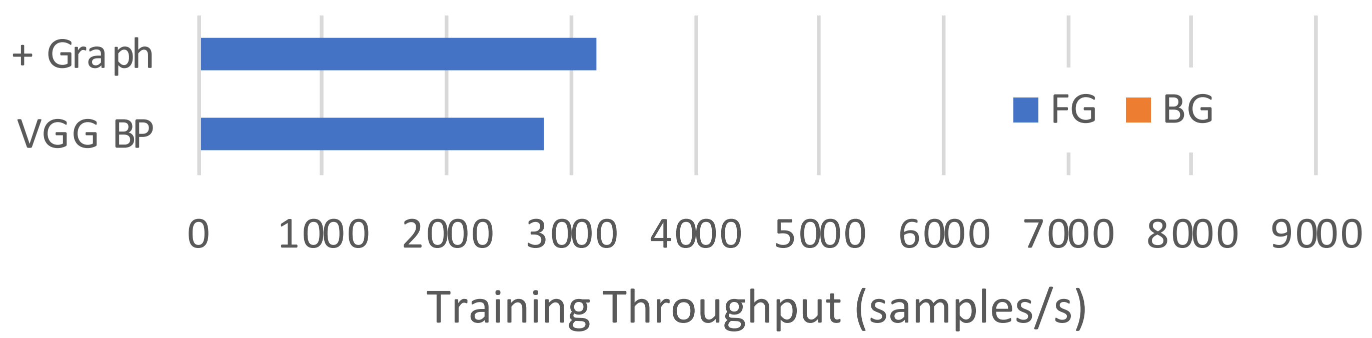


(b) WideResNet-101-2, scaling b=16



(c) InceptionV3, scaling b=32

Decomposition of Each QoS Techniques



Multiplexing VGG16 on a cluster with 8x A100 GPUs.

Conclusion

- **Two techniques for efficiently scaling DNN training:**
  1. Burst parallel training
  2. GPU multiplexing
- **Limitations**
  - Strong scaling only on the sample dimension & parallel layers
  - Background jobs run on a single GPU

Questions?

https://github.com/seojinpark/DeepPool

<seojin@csail.mit.edu>