Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning

Aurick Qiao[1,2], Sang Keun Choe[2], Suhas Jayaram Subramanya[2], Willie Neiswanger[1,2], Qirong Ho[1], Hao Zhang[1,3], Gregory R. Ganger[2], Eric P. Xing[4,1,2]

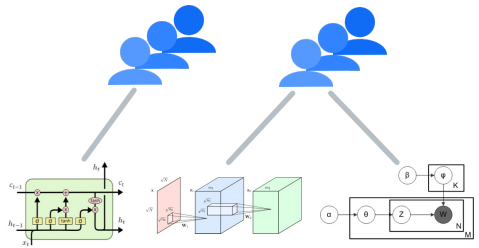
[1]Petuum Inc., [2]Carnegie Mellon University, [3]UC Berkeley, [4]MBZUAI

Aurick Qiao

OSDI'21
7/14/2021

Deep Learning Training in Shared Clusters

Many users and training jobs



Time/compute-intensive

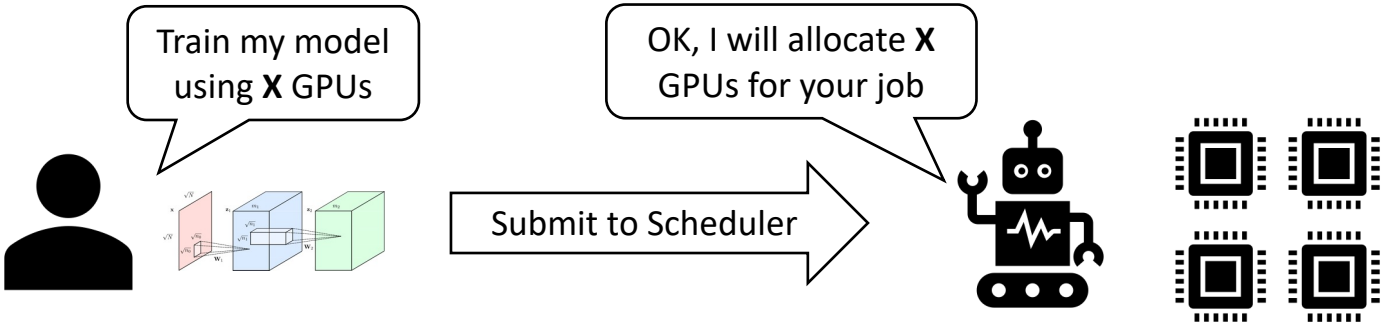
Scheduler

Shared compute cluster

Expensive hardware (e.g., GPUs)

Cluster scheduler decides how to allocate resources to jobs in order to minimize **training time**, maximize **cluster utilization**, or ensure **fairness**.

Example Shared-Cluster DL Training Workflow



How to determine number of GPUs **X**?
Depends on:

- **Cluster contention**: changes dynamically
- **Job scalability:** needs expert knowledge

Inter-dependent, dynamic decisions

How to configure training parameters to utilize the allocated GPUs efficiently?

- Tune **batch size** and **learning rate**.

Not managed by existing cluster schedulers for DL training!

Pollux: Co-adaptive Cluster Scheduler for DL

Automatically and dynamically:

A. Allocates **resources** considering cluster-wide performance and fairness.

B. Tunes the **batch size** and **learning rate** for each individual training job.

Co-adaptive:

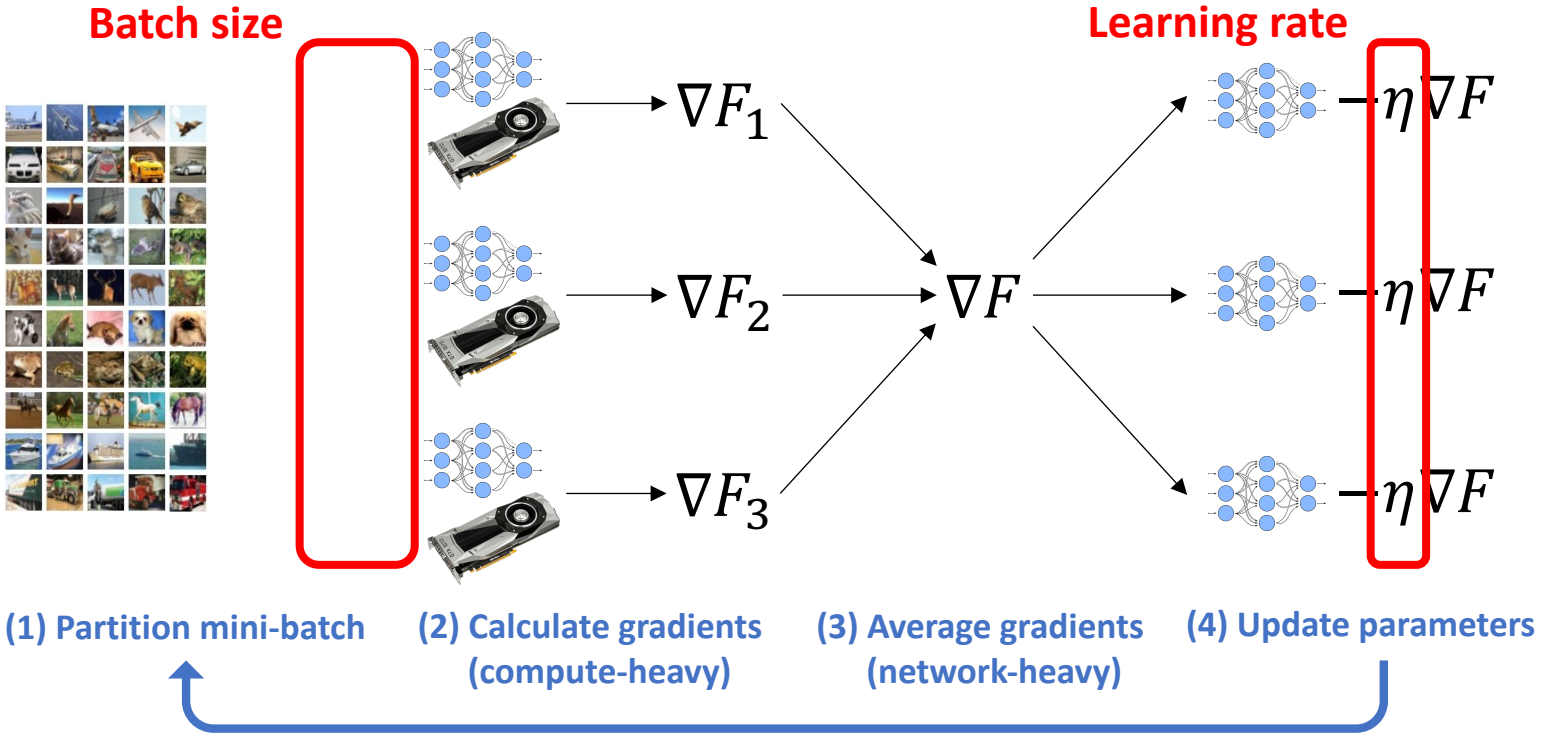
- Cluster-wide scheduling decisions account for the tuning happening at the per-job level.
- Individual jobs are tuned adaptively to the resources allocated by the cluster scheduler.

Results: reduces manual job configuration, improves average training time by **37-50%**.

Outline

1. Motivation
2. Background on DL Training
3. Design of Pollux
4. Evaluation Results

Background: Distributed DL (Data Parallelism)

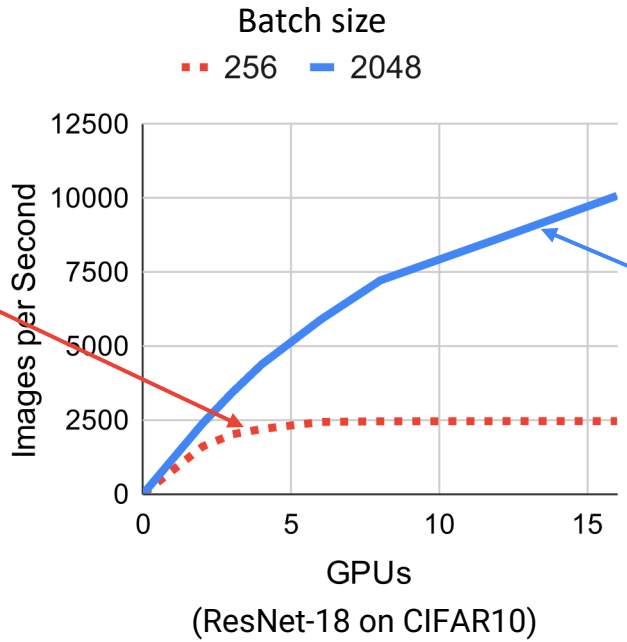


System Throughput and Impact of Batch Size

DL training scales **sub-linearly**: too many GPUs doesn't help to increase system throughput!

Hard to know how many GPUs to use, depends on:

- Model architecture
- Cluster hardware
- Training batch size
- …



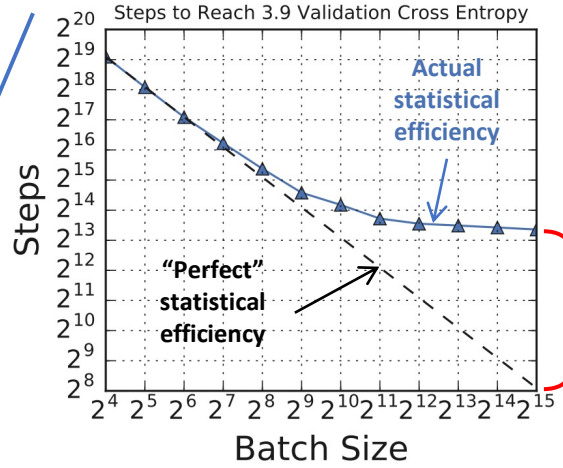
(ResNet-18 on CIFAR10)

Common strategy:
Use a larger batch size to improve system throughput/scalability

Statistical Efficiency and Impact of Batch Size

Why not keep increasing the batch size?

1. Need to carefully tune learning rate to keep the model quality consistent.
2. Increasing the batch size decreases the **statistical efficiency** of DL training.
3. Even further increasing the batch size results in worse model generalization.



(f) Transformer on LM1B

(Shallue et al. 2018)

Illustration of Overall Training Performance

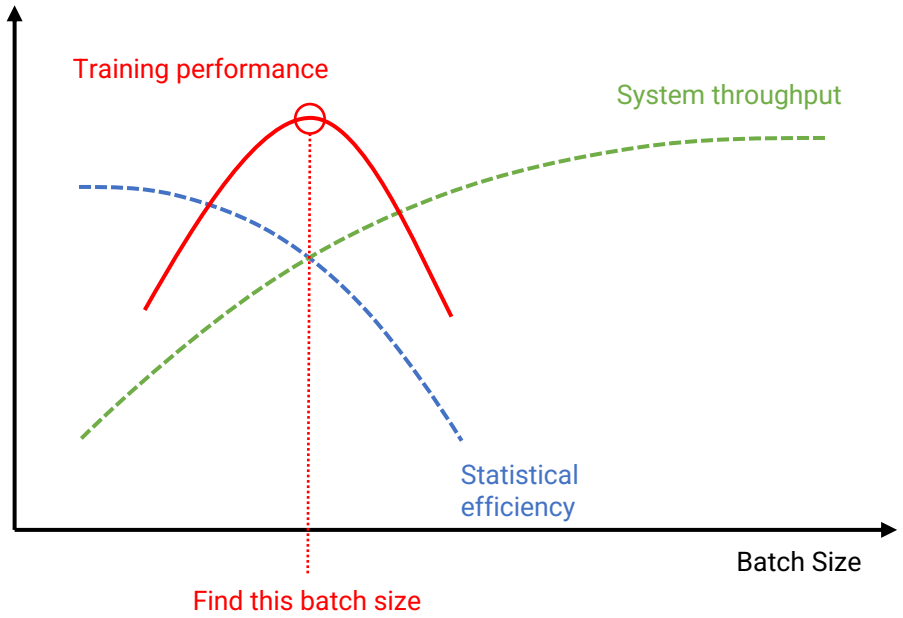
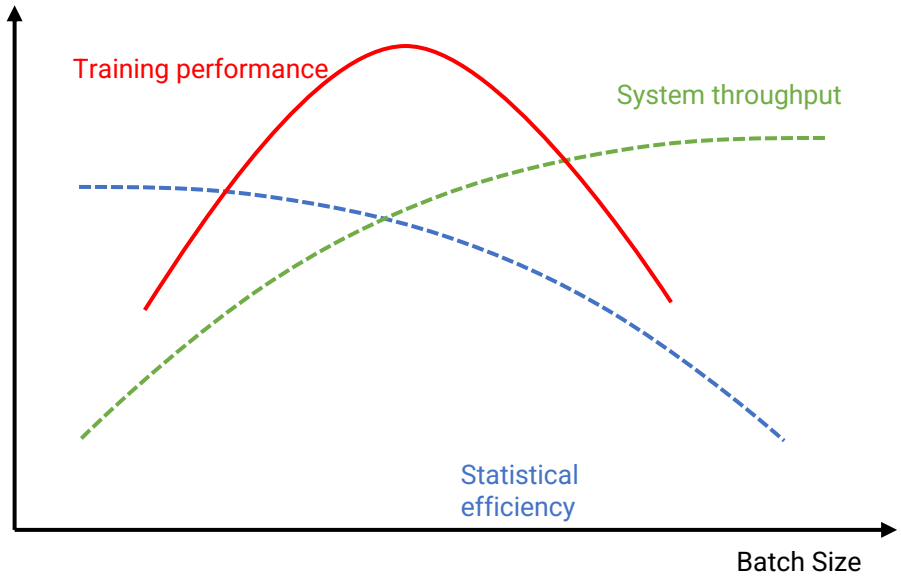


Illustration of Overall Training Performance



Statistical efficiency increases during training
→ Optimal batch size changes dynamically!
(by 10x or more, McCandlish et al. 2018)

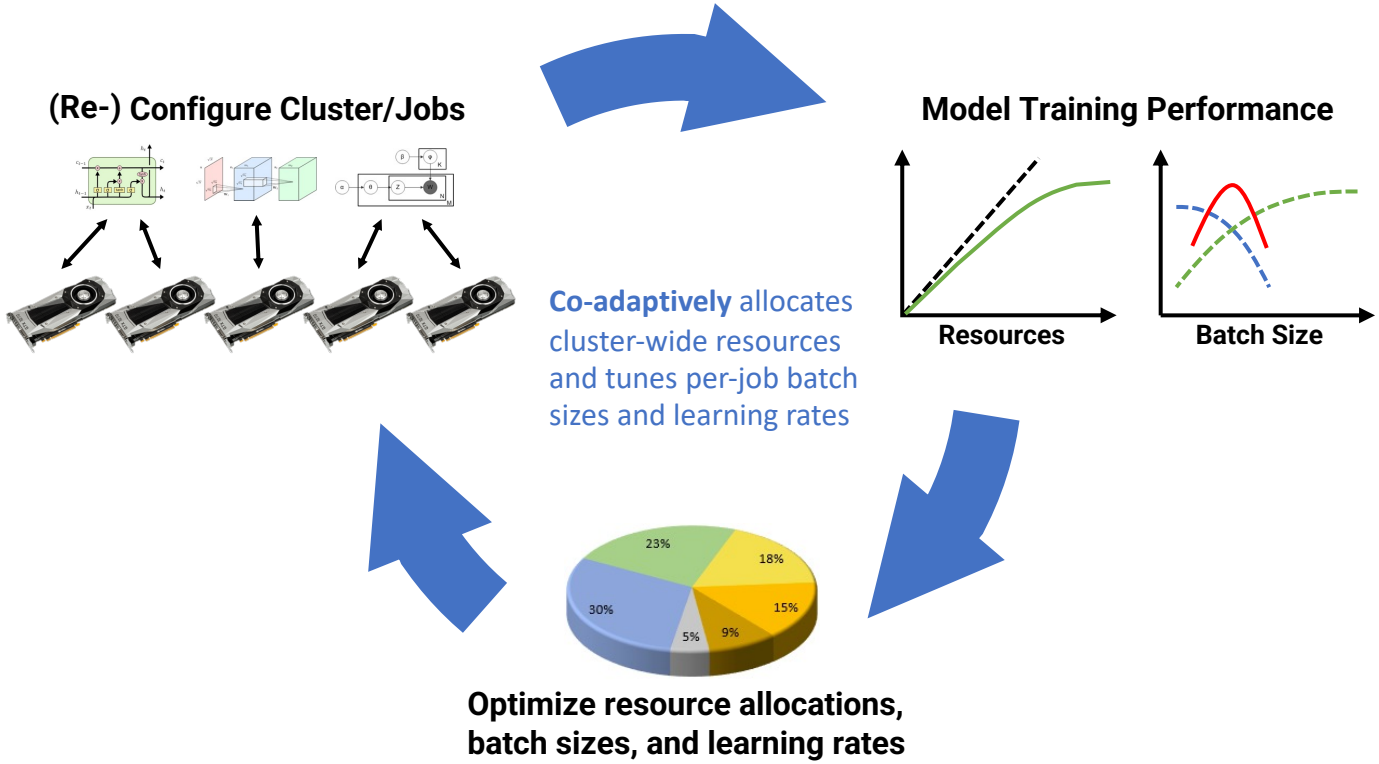
Implications for Cluster Scheduling

1. The preferred GPU allocation for a DL training job depends on its batch size, while the preferred batch size of the job depends on its allocated GPUs **(inter-dependency)**.
2. GPU allocation also depends on cluster-wide factors such as fairness and contention.
3. Batch size also depends on per-job factors such as scalability and statistical efficiency.

Hard for users to account for these inter-dependent factors when submitting jobs.

A cluster scheduler that jointly controls these factors can better optimize for DL training.

Pollux Cluster Scheduler



Key Idea: Goodput, not Throughput

Pollux optimizes for a new measure of DL training performance called the *goodput*.

$$\text{GOODPUT}_t(a, m, s) = \text{THROUGHPUT}(a, m, s) \times \text{EFFICIENCY}_t(M)$$

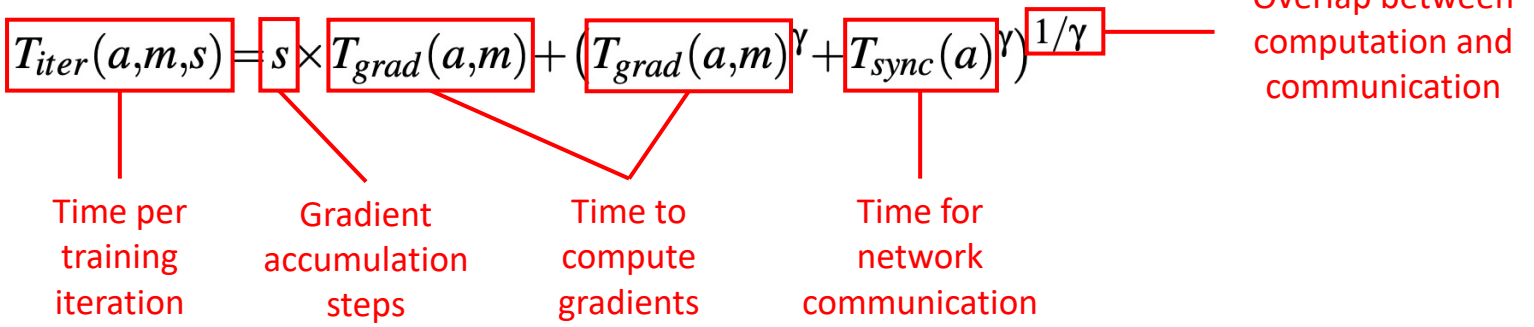
***System* throughput**
(training examples / second)

***Statistical* efficiency**
(progress / training example)

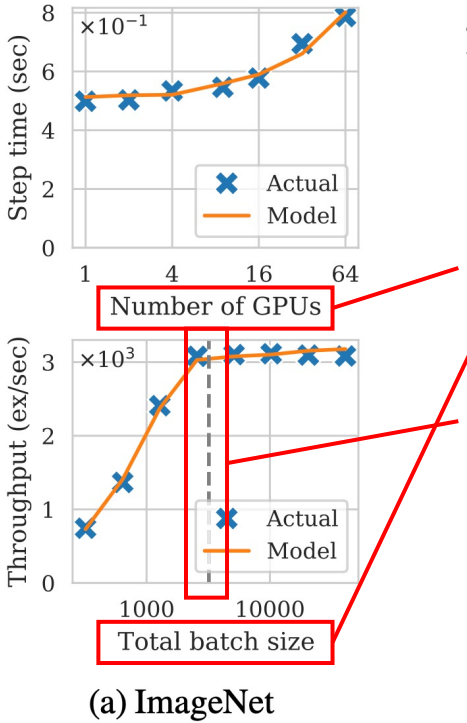
Automatically determined by Pollux for each job during training

- $\boldsymbol{a}$: Allocation vector, $\boldsymbol{a_n}$ = #GPUs on node $\boldsymbol{n}$
- $\boldsymbol{m}$: Per-GPU batch size
- $\boldsymbol{s}$: Gradient accumulation steps (enables total batch sizes larger than GPU memory limit)
- $\boldsymbol{M}$: Total batch size, $\boldsymbol{M} = |\boldsymbol{a}| \times \boldsymbol{m} \times \boldsymbol{s}$

Modeling System Throughput



Modeling System Throughput



(a) ImageNet

$$T_{iter}(a,m,s) = s \times T_{grad}(a,m) + \left(T_{grad}(a,m)^{\gamma} + T_{sync}(a)^{\gamma}\right)^{1/\gamma}$$

Accurately model observed throughput for various numbers of GPUs and batch sizes.

Accurately model the effect of accumulating gradients locally without synchronization.

Accurately model the effect of GPUs being allocated on different nodes vs same node.

Enables Pollux to automatically:

1. Determine the right # GPUs and batch size.
2. Use gradient accumulation to increase the batch size beyond the limits of GPU memory.
3. Try to pack a job's GPUs onto fewer nodes to minimize network overhead.

Modeling Statistical Efficiency

Pollux also models the statistical efficiency for each DL training job:

$$\texttt{EFFICIENCY}_t(M) = \frac{\varphi_t + M_0}{\varphi_t + M}$$

User provides a static baseline batch size M_0

EFFICIENCY of batch size M is measured relative to M_0

User can simply pick a small M_0 and not worry about system throughput or scalability.

Rely on Pollux to pick different M that balances system throughput and statistical efficiency.

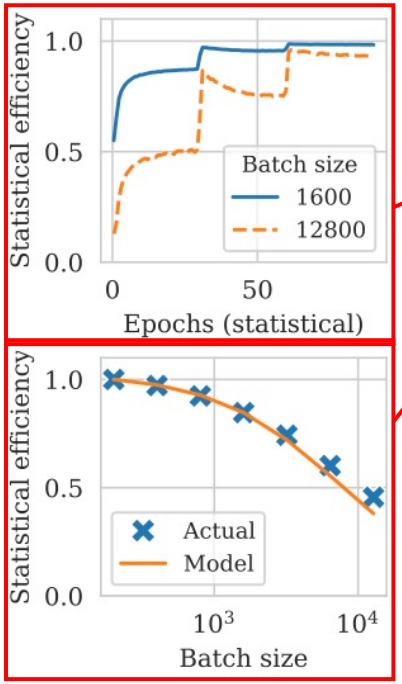
φ_t = (Pre-conditioned) Gradient Noise Scale

(McCandlish et al. 2018)

Gradient noise scale intuition:

- Higher gradient noise → larger mini-batch is helpful → higher statistical efficiency.
- Lower signal-to-noise near convergence → better statistical efficiency later in training.

Modeling Statistical Efficiency



(a) ImageNet

$$\texttt{EFFICIENCY}_t(M) = \frac{\varphi_t + M_0}{\varphi_t + M}$$

Lower observed EFFICIENCY for larger batch sizes, improves later in training.

Accurately predict what EFFICIENCY would be if using a different batch size.

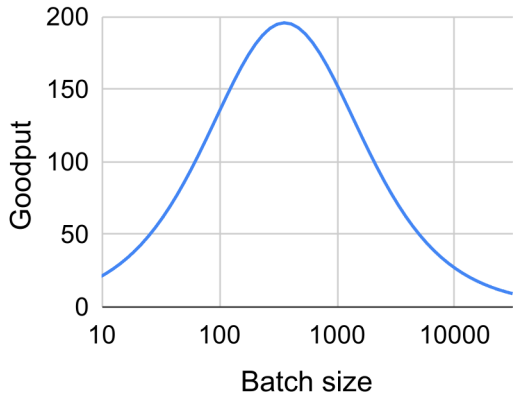
Putting it all together:

$$\text{GOODPUT}_t(a, m, s) = \text{THROUGHPUT}(a, m, s) \times \text{EFFICIENCY}_t(M)$$

Pollux quickly predicts what the GOODPUT of a training job would be using (a, m, s) without needing to run the training job using (a, m, s).

Pollux Optimizes Goodput for Each DL Job

Given an allocation of GPUs a, solve: $m^*, s^* = \underset{m,\, s}{\operatorname{argmax}} \operatorname{GOODPUT}_t(a, m, s)$



(ResNet-18 on CIFAR10 with a and s fixed to a constant)

Changing the batch size means the learning rate must be re-tuned.

Pollux lets user select a suitable *learning rate scaling rule* established in the DL community:

- Linear scaling
- Square-root scaling
- AdaScale (Johnson et al. 2020)
- …

Optimizing Cluster-Wide Allocations

Pollux finds an *allocation matrix* A, where A_{jn} = #GPUs on node n allocated to job j.

Optimization objective:

$$\texttt{FITNESS}_p(A) = \left(\frac{1}{J}\sum_{j=1}^{J}\texttt{SPEEDUP}_j(A_j)^p\right)^{1/p}$$

p is a tunable parameter that controls fairness between jobs.

where

$$\texttt{SPEEDUP}_j(A_j) = \frac{\max_{m,s}\texttt{GOODPUT}_j(A_j,m,s)}{\max_{m,s}\texttt{GOODPUT}_j(a_f,m,s)}$$

Best goodput with given allocation A_j

Best goodput with fair-share of GPUs (normalize across jobs)

Accounts for job-level tuning!

Search for A with a metaheuristic algorithm.

- Pollux uses a Genetic Algorithm.

Additional scheduling considerations:

- Avoid frequent re-allocations (high overhead).
- Avoid distributed jobs sharing a node (interference).

Evaluation of Pollux

A primary benefit of Pollux is automatically configuring jobs in shared clusters.

Important evaluation objective: demonstrate that even if jobs are given ideal static configurations, Pollux still outperforms alternative DL cluster schedulers.

- Real-world job distributions from Microsoft's DL cluster traces (Jeon et al. 2019).
- Mix of training tasks: image classification, object detection, speech recognition, question answering, and recommendation.
- Manually tuned #GPUs, batch size, learning rate, and gradient accumulation ahead of time
  - For setting up strong baselines, does not benefit Pollux!

Scheduling Expert-Configured Jobs

16 nodes w/ 4 GPUs each (Nvidia T4)

160 DL jobs submitted over 8 hours

Policy	Job Completion Time		Makespan
	Average	99%tile	
Pollux ($p=-1$)	**0.76h**	11h	**16h**
Optimus+Oracle+TunedJobs	1.5h	15h	20h
Tiresias+TunedJobs	1.2h	15h	24h

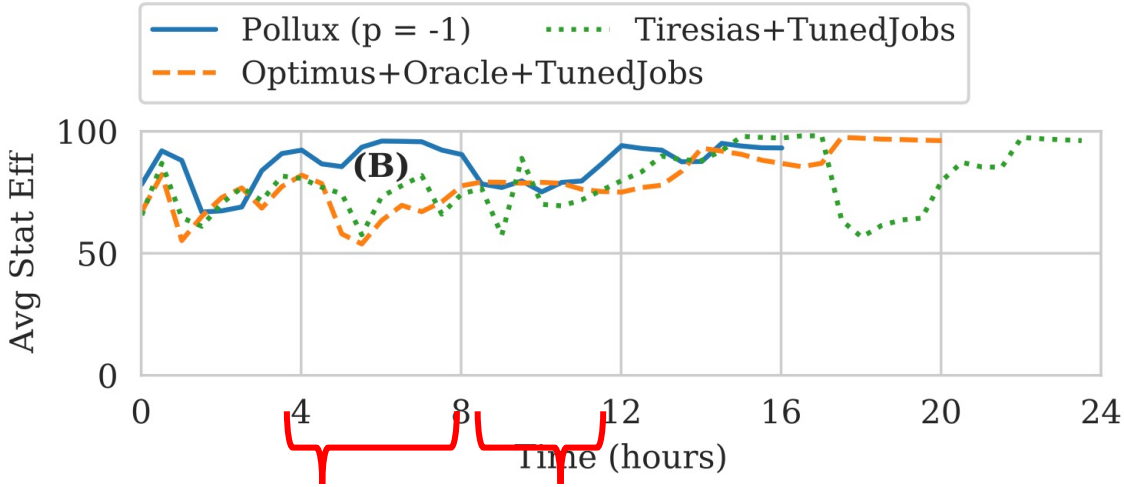
Uses expert-configured jobs.
Can elastically adapt resources, but not batch size/learning rate. (Peng et al. 2018)

Uses expert-configured jobs.
Can pause/resume jobs based on their GPU-time metrics. (Gu et al. 2019)

37-50% faster average training time

More realistic job configs:
>70% faster average training time

Cluster-wide Statistical Efficiency



Period of **high** cluster contention:

Pollux reduces #GPUs and batch size and trains with higher statistical efficiency.

Period of **low** cluster contention:

Pollux accepts lower statistical efficiency to train with higher system throughput.

Key trade-off made by Pollux

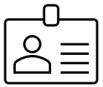
More Experiments in our Paper!

- Scheduling fairness
- Sensitivity studies
- Hyper-parameter tuning workloads
- Improving auto-scaling in the cloud
- …

Conclusion

- Pollux co-optimizes both cluster-wide and per-job parameters for DL training.
- Pollux introduces goodput, a measure of training performance that combines system throughput and statistical efficiency.
- Pollux improves average training time in shared clusters by 37-50%, even against unrealistically strong baselines.

Open-sourced at https://github.com/petuum/adaptdl.



aqiao@andrew.cmu.edu