Elastic Resource Sharing for Distributed Deep Learning

Changho Hwang,
Taehyun Kim, Sunghyun Kim*, Jinwoo Shin, and KyoungSoo Park

KAIST, MIT*

KAIST
NDsl
Networked and Distributed Computing Systems Lab
Multi-tenant Training Workload

• Very expensive workload
  • Deep learning training (DLT) is costly
    • Roughly, $1=1K$ params training [1]
  • Users often write jobs that scale poorly
  • Many parallel DLT jobs

Need an efficient system that achieves:
✓ Low average job completion time (avg. JCT)
✓ Fairness between jobs
✓ High resource utilization

Efficient Resource Sharing

• **Resource sharing**: key to efficient cluster usage

  *non-elastic sharing vs. elastic sharing*
Non-elastic Resource Sharing

- Each job declares/requests its own resource share
- *Job-level coarse-grained scheduling*
Elastic Resource Sharing

- System determines the share of every job
- Resource-level fine-grained scheduling

Good for optimization, but realizing the benefit is challenging!
Our Work: Algorithms + Systems

1. **Apathetic Future Share (AFS):** elastic sharing for reducing avg. JCT

2. **CoDDL:** resource management system optimized for elastic sharing
Our Work: Algorithms + Systems

1. Apathetic Future Share (AFS): elastic sharing for reducing avg. JCT

2. CoDDL: resource management system optimized for elastic sharing
Reducing Average Job Completion Time

• Prioritize short jobs!

The SJF scheduling algorithm is provably optimal, in that it gives the minimum average waiting time for a given set of processes. Moving a short process before a long one decreases the waiting time of the short process more than it increases the waiting time of the long process. Consequently, the average waiting time decreases.

– Operating System Concepts, 8th ed.

• Use shortest-remaining-time-first (SRTF, or preemptive SJF)

No, SRTF is often suboptimal for DLT jobs!
Motivating Questions

• What if the shortest job's throughput does not scale?
  → No more throughput, no more resources! (Of course.)

• What if it is somewhere in between?
  → Non-trivial
## Handling Shortness & Efficiency

- **Resource efficiency** introduced in the picture!

### Algorithms Side

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Prioritize short job</th>
<th>Prioritize efficient job</th>
<th>Elastic Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRTF</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Max-min</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Optimus [EuroSys '18]</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SRSF/Tiresias [NSDI '19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Themis [NSDI '20]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AFS (this work)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- ✓: handled explicitly
- △: handled implicitly
AFS Resource Sharing

• Strike a balance between shortness & efficiency
  • Say that job $b$ remains longer than job $a$, then:
    \[
    \frac{p'_b - p_b}{p'_b} - \frac{p'_a - p_a}{p_a} \begin{cases} > 0, & \text{prioritize } b \text{ over } a \text{ (efficiency wins)} \\ \leq 0, & \text{prioritize } a \text{ over } b \text{ (shortness wins)} \end{cases}
    \]
  
  \(p_x\): current throughput of job $x$
  \(p'_x\): throughput of job $x$ with one more GPU

• What does this mean?
  • At first, do not bias into either side
  • **Lighter** resource contention \(\rightarrow\) bias towards **short** jobs
  • **Heavier** resource contention \(\rightarrow\) bias towards **efficient** jobs

Refer to rigorous analysis in our paper!
AFS Scheduling Algorithm – AFS-L

• Leverages the exact remaining time, like SRTF
  • For apple-to-apple comparison with SRTF
• Real-world trace evaluation (simulated)
AFS Scheduling Algorithm – AFS-P

- Can run without job length information
  - Leverage processor sharing
  - More practical than AFS-L
- Real-world trace evaluation (simulated)

![Graph showing average JCT reduction rate for different traces with bars for Tiresias-L, Max-min, Optimus, Themis, and AFS-P.]

More details of algorithms in the paper!
Overview

Our Work: Algorithms + Systems

1. **Apathetic Future Share (AFS):** elastic sharing for reducing avg. JCT

2. **CoDDL:** resource management system optimized for elastic sharing
Realizing Elastic Resource Sharing

- Frequent job reconfigurations drop the performance
  - **6.3x on average, & up to 22x** more than non-elastic sharing
  - Severe thrashing on a burst of reconfigurations
Experiment: Cluster Thrashing

- Replay a full-scaled trace in a real 64-GPU cluster
- Use AFS-P scheduler
- Monitor a single job submitted at time zero

![Graph showing frequent job submission and very slow progress](image)
CoDDL: Coordinator for Distributed Deep Learning

• Overview
  • Front-end: accepts user-written DL models
  • Back-end: run the scheduler & auto-scale models

• Minimize the job reconfiguration overhead
  1. Concurrent reconfiguration with job execution
  2. Fast cancelling stale configuration commands

See other system details in the paper: failure handling, GPU placement, throughput measurement, etc.
Experiment: Relaxed Thrashing

- Accelerate job progress by relaxing thrashing

Find avg. JCT evaluation over CoDDL in the paper!
Summary

• Elastic resource sharing for multi-tenant DLT jobs
  • Enables significant improvement of scheduling performance

• Apathetic Future Share (AFS): elastic resource sharing algorithm
  • Balanced prioritization of short jobs & efficient jobs for reducing avg. JCT
  • Reduce avg. JCT up to 2.67x and 3.11x over SRTF and Tiresias-L, respectively

• CoDDL: optimized resource management system for elastic sharing
  • Mitigate cluster thrashing via efficient job reconfiguration