Balancing Efficiency and Fairness in Heterogeneous GPU Clusters for Deep Learning

Shubham Chaudhary | Ramachandran Ramjee | Muthian Sivathanu
Nipun Kwatra | Srinidhi Viswanatha

Microsoft Research India
## Scheduling of Deep Learning Workloads

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Exclusive GPU</th>
<th>Execution Model</th>
<th>Optimizes For</th>
<th>Fairness</th>
<th>Heterogeneity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FfDL¹</td>
<td>✔️</td>
<td>Generic</td>
<td>Scalability</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Philly²</td>
<td>✔️</td>
<td>Generic</td>
<td>Consolidation</td>
<td>×</td>
<td>Static Partitioning + Preemption</td>
</tr>
<tr>
<td>Optimus³</td>
<td>×</td>
<td>Parameter Server</td>
<td>Average JCT*</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Tiresias⁴</td>
<td>×</td>
<td>Parameter Server</td>
<td>Average JCT*</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Gandiva⁵</td>
<td>×</td>
<td>Generic</td>
<td>Utilization</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>


* Job Completion Time
Performance Isolation and Fair Share

• How to share a large cluster among many different groups?
• **Simple:** Perform static partitioning of a physical cluster into virtual clusters.
  • Makes sharing of underutilised resources hard.
• **Idea:** Provide performance isolation through proportional allocation of resources.
• New GPUs released each year.
• Separate physical clusters for each generation, users choose which cluster to submit to.
• Everyone wants newer GPUs, therefore older GPUs left underutilized.
• How to choose the best GPU automatically?
Contributions

Gandiva\textsubscript{fair} is the first Deep Learning Scheduler that does

\begin{itemize}
  \item Efficient fair-sharing of cluster-wide GPU throughput.
  \item Transparent handling of resource heterogeneity.
  \item Migration to provide the above without preemption.
\end{itemize}

\textit{One cluster scheduler to rule them all.}
System Model

• Users are assigned tickets and GPU throughput is allocated proportionally among all active users.
• Tickets are divided equally among all jobs of the same user.
• Jobs can be of varying sizes, GPUs should be gang-scheduled.
• We use the time-slicing and migration primitives implemented in Gandiva\(^5\).
/** called every time-quantum. */

def schedule:
    job = min(q, λj: j.pass)
    job.pass += 1 / job.tickets
    return {job}
Split-Stride Scheduler

**Gang-Aware Stride Scheduling**

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>A, B, C</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>A, B, D</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>A, B, C</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>A, B, D</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>A, B, C</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>A, B, D</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>A, B, C</td>
</tr>
</tbody>
</table>

```python
/* called every time-quantum. */
def schedule:
    freeGPUs = numGPUs
    scheduled = {}
    jobs = sort(q, λj: j.pass)
    i = 0
    while freeGPUs > 0 and i < length(jobs):
        if jobs[i].size ≤ freeGPUs:
            scheduled U= {jobs[i]}
            freeGPUs -= jobs[i].size
            jobs[i].pass += jobs[i].size / jobs[i].tickets
    return scheduled
```

<table>
<thead>
<tr>
<th>Job</th>
<th>Tickets</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Split-Stride Scheduler

- **Simple**: Run Gang-Aware Stride across all GPUs on a cluster.
  - Not scalable and unbounded migrations.

- **Idea**: Run a Gang-Aware Stride locally on each server.
  - How to run multi-server jobs? Some central coordination is required.
Split-Stride Scheduler

Schedule is fair if the load is balanced across all servers.

[6] Refer to the paper for details.
Handling GPU Heterogeneity

- Transparently profile jobs to determine speedups on all GPU generations
- Assumption: each user submits the same type of job.
- For example, as a part of hyperparameter exploration.
- Place jobs on the fastest GPU subject to contention.

<table>
<thead>
<tr>
<th>Job</th>
<th>K80 (ms)</th>
<th>K80 / P40</th>
<th>K80 / P100</th>
<th>K80 / V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAE</td>
<td>11.5</td>
<td>1.17</td>
<td>1.19</td>
<td>1.25</td>
</tr>
<tr>
<td>SuperResolution</td>
<td>207.5</td>
<td>1.43</td>
<td>1.73</td>
<td>1.87</td>
</tr>
<tr>
<td>DCGAN</td>
<td>183.4</td>
<td>4.34</td>
<td>4.31</td>
<td>6.42</td>
</tr>
<tr>
<td>GRU</td>
<td>48.4</td>
<td>3.00</td>
<td>2.58</td>
<td>4.81</td>
</tr>
<tr>
<td>LSTM</td>
<td>48.9</td>
<td>3.10</td>
<td>3.58</td>
<td>4.81</td>
</tr>
<tr>
<td>ResNet50</td>
<td>134</td>
<td>3.17</td>
<td>3.34</td>
<td>5.14</td>
</tr>
<tr>
<td>ResNeXt50</td>
<td>2005.7</td>
<td>3.70</td>
<td>4.12</td>
<td>6.33</td>
</tr>
</tbody>
</table>
Automated Resource Trading

- **Idea:** If we exchange U1’s 1 V100 for U2’s $p$ K80s, both users gain if $1.2 < p < 6$.

- For maximum efficiency gain, trade between highest and lowest speedup users.

- **Issue:** user gaming, for example, user artificially slows down their K80 jobs to win V100s.

- **Idea:** Use speedup as bids in a Vickrey auction, $p$ as second-price is *incentive-compatible*. For example, if another user U3 exists with a 2X speedup, then $p$ is 2.
Implementation

- Implemented as a custom scheduler on Kubernetes.
- Manager contacts the Gandiva Client to perform operations like time-slicing.
Fair-Share on a Homogeneous Cluster

- Each user obtains close to their fair share.
  - 48 P100 GPU cluster.
  - 70 users with one 1, 2, 4 or 8 GPU jobs with job size distribution derived from Philly Trace\textsuperscript{2,7}.

Benefit of Trading on Heterogeneous Cluster

- Users 1 and 4 exhibit about 30% increase in performance.
- Users 2 and 3 exhibit similar performance.

- 100 GPU cluster with 12 V100s, 24 P100s, and 128 K80s.
- 4 users with many 1, 2, or 4 GPU jobs with different speedups.
Summary

• Gandiva_{fair} is a domain specific scheduler for Deep Learning workloads.
• Provides efficient fair-sharing of cluster-wide GPU throughput among users.
• Handles heterogeneous GPUs transparently using profiling and automated resource trading.