Gandiva: Introspective Cluster Scheduling for Deep Learning

Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, Lidong Zhou

Microsoft Research
Deep learning: An important cloud workload

• Growing impact: Consumer products – Web search, Alexa/Siri/Cortana, ...
  • Upcoming: Enterprise uses (e.g. medical diagnosis, retail)

• DL jobs are compute-intensive, so need expensive custom hardware
  • Dominant platform today: GPUs
  • Cloud vendors run large clusters of GPUs (billions of $)

• Efficient use of GPU clusters crucial to manage cost of DL innovation
Deep Learning Training (DLT)

• Build a model for an end-to-end application (e.g. speech2text)
  • Select best model architecture, invent new architectures, tune accuracy, ...
  • Key to DL Innovation

• DLT is mostly trial-and-error: Little theoretical understanding
  • Will a model architecture work? Don’t know -- Train it and measure!
  • Lots of trials => high cost: Training = significant fraction of GPU usage

• Goal: Run DLT jobs efficiently in a cluster of GPUs
DLT Schedulers today

• Treat DLT jobs as generic big-data jobs (e.g. use Yarn, Kubernetes)
• Schedule a job on a GPU **exclusively**, job holds it **until completion**
• Problem #1: **High Latency** (head-of-line blocking)

**Need time-slicing of jobs**

However, GPUs not efficiently virtualizable
DLT Schedulers today

- Treat DLT jobs as generic big-data jobs (e.g. use Yarn, Kubernetes)
- Schedule a job on a GPU **exclusively**, job holds it **until completion**
- Problem #2: **Low Efficiency** (Fixed decision at job-placement time)

![Diagram showing two servers, one with a 2-GPU job, and sensitivity to locality variations across jobs.]
Domain knowledge: Intra-job predictability

Each spike is a “mini-batch”

Mini-batch times identical

~77x diff. in RAM usage

Time-slicing quantum = Group of minibatches

ResNet50 training on ImageNet data
**Gandiva: A domain-specific scheduler for DLT**

- **Result**: Faster & cheaper execution of DLT workflows
  - **Latency**: 4.5x lower queueing times, 5-7x faster multi-jobs (AutoML)
  - **Efficiency**: 26% higher cluster throughput
Outline

• *Introduction*
• Gandiva mechanisms
• Implementation & Evaluation
• Conclusion
Time-slicing

- Over-subscription as a first-class feature (similar to OS)
  - Time quantum of ~1 min (~100 mini-batches)
  - Better than queueing: Faster time-to-early feedback
  - Faster multi-job execution during hyper-param searches

**Customization:** Align with mini-batch boundary => ~50x cheaper
Migration / Packing

• Move jobs across GPUs to improve efficiency
• Generic distributed process migration is unreliable / slow
  • Customization: Integration with toolkit checkpointing makes it fast/robust

• #1: De-fragment multi-GPU jobs
• #2: Exploit heterogeneity: Low job parallelism => cheaper GPU
• #3: Packing: Pack multiple jobs onto the same GPU
  • Jobs that are low on GPU & RAM usage. Run together instead of time-slice

• Challenge: How do we know migration/packing helped?
Application-aware profiling

Two possibilities:
- #1: 30% more useful work done
- #2: Overhead due to interference
  - Could even be a net loss!

• Solution: Measure useful work directly
  • Customization: Job runtime exports “time-per-minibatch”

• Allows simple “introspection” policy
  • Try migration/packing, measure benefit, revert if negative
### Introspective Scheduling

<table>
<thead>
<tr>
<th></th>
<th>Traditional Schedulers</th>
<th>Gandiva</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scheduling</strong></td>
<td>One-time (job-placement)</td>
<td>Continuous / Introspective</td>
</tr>
<tr>
<td>decision</td>
<td>- Stuck with decision for entire job</td>
<td>- Can recover quickly from mistakes</td>
</tr>
<tr>
<td><strong>Profiling</strong></td>
<td>System-level: e.g. CPU/GPU Util</td>
<td>Application-level (<em>customized</em>): Mini-batches per second</td>
</tr>
<tr>
<td></td>
<td>- Entangles Useful work vs. overhead</td>
<td>- Measures “useful work”</td>
</tr>
</tbody>
</table>
Outline

• Introduction
• Schedulers for DLT: Today
• Gandiva mechanisms
• Implementation & Evaluation
• Conclusion
Implementation

Gandiva Scheduler

Time_Slice()
Do_Migration()
Do_Packing()

Profile / Job State

Node / Container Info
Node allocation req.

Kubernetes Master
Kubernetes API

Job creation / Node allocation

Kubernetes Node
Kube Daemon

Gandiva Client

Start, Stop, Pause, Resume,…

Profile Info / Job State

Scheduling RPCs

Container

User DLT Job

Also, changes to DL Toolkits: Tensorflow & pyTorch

Time-slicing, migration, etc.
Microbenchmark: Time-slicing

Server 4 P100 GPUs

6 DLT jobs: ResNet50/ImagNet on pyTorch

All jobs get equal time-share during time-slicing

Low overhead: Total throughput remains same
Micro-benchmark: Packing

1 P100 GPU
2 DLT jobs: Image Superresolution on pyTorch

Gandiva starts with time-slicing

Based on profiling, tries to pack both jobs

Higher App throughput ➞ Continue w/ packing
Microbenchmark: AutoML

AutoML: Explore 100 hyper-parameter configs
- ResNet-like Model for CIFAR Image dataset; 16 P40 GPUs
- HyperOpt: Predict “more promising” mode based on early feedback

Time-slicing + Prioritization => Gandiva explores more configs in parallel

<table>
<thead>
<tr>
<th></th>
<th>Accuracy: 70%</th>
<th>Accuracy: 80%</th>
<th>Accuracy: 90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>134.1</td>
<td>2489.1</td>
<td>5296.7</td>
</tr>
<tr>
<td>Gandiva</td>
<td>134.1</td>
<td>543.1</td>
<td>935.4</td>
</tr>
<tr>
<td>Speedup</td>
<td>1x</td>
<td>5.25x</td>
<td>5.66x</td>
</tr>
</tbody>
</table>

Time in minutes to find config w/ accuracy > threshold
Cluster utilization

Cluster of 180 GPUs

Synthetic DLT jobs modelled from a production trace

Efficiency
Cluster throughput improves by 26%

Latency
4.5x reduction in avg. time to first 100 mini-batches
Summary

• Large cloud applications benefit from custom systems infrastructure
• Co-design of cluster scheduler w/ DL job => rich information, control

• Efficient time-slicing => Low latency, early feedback, iterate fast
• Application-aware profiling => Introspection
• Custom migration/packing => Cluster efficiency
• Much faster hyper-parameter exploration/AutoML