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

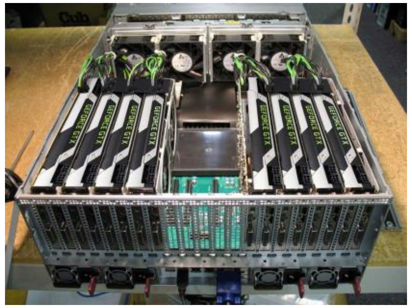
- DLT is mostly **trial-and-error**: Little theoretical understanding
  - Will a model architecture work? *Don’t know -- Train it and measure!*
  - Lots of trials => high cost: Training = significant fraction of GPU usage

- **Goal: Run DLT jobs efficiently in a cluster of GPUs**

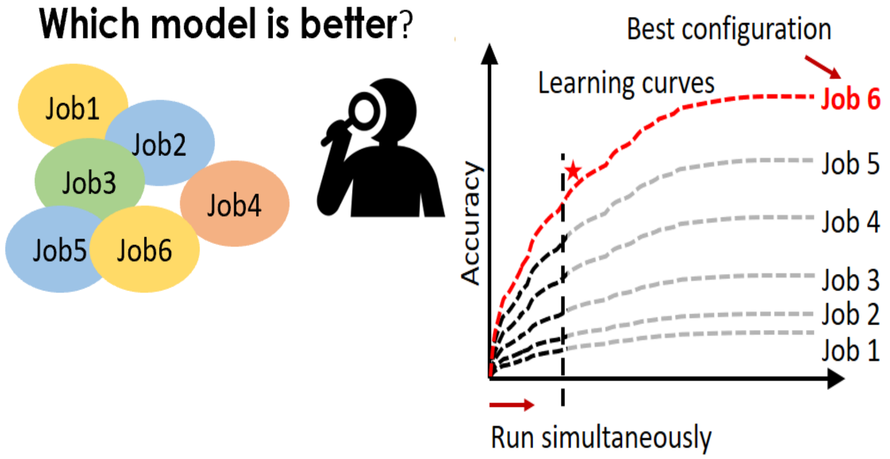
DLT Schedulers today

- 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)

Short job
(queued)



Long DLT job
Runtime: Several days!



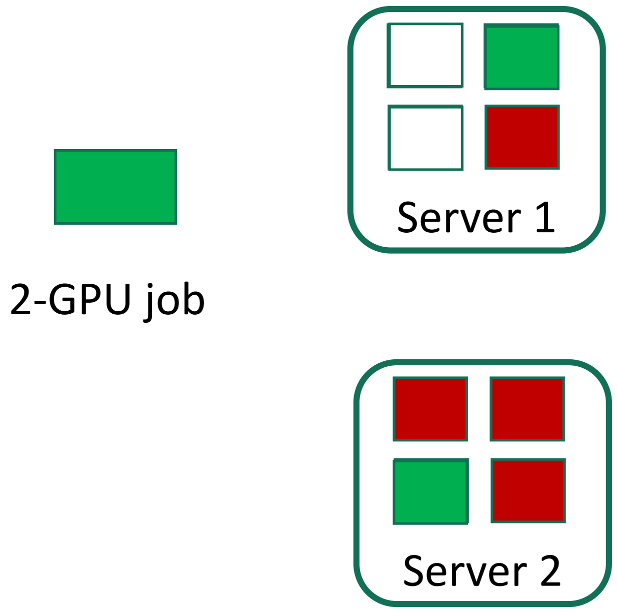
Need time-slicing of jobs

Multi-job

However, GPUs not efficiently virtualizable

DLT Schedulers today

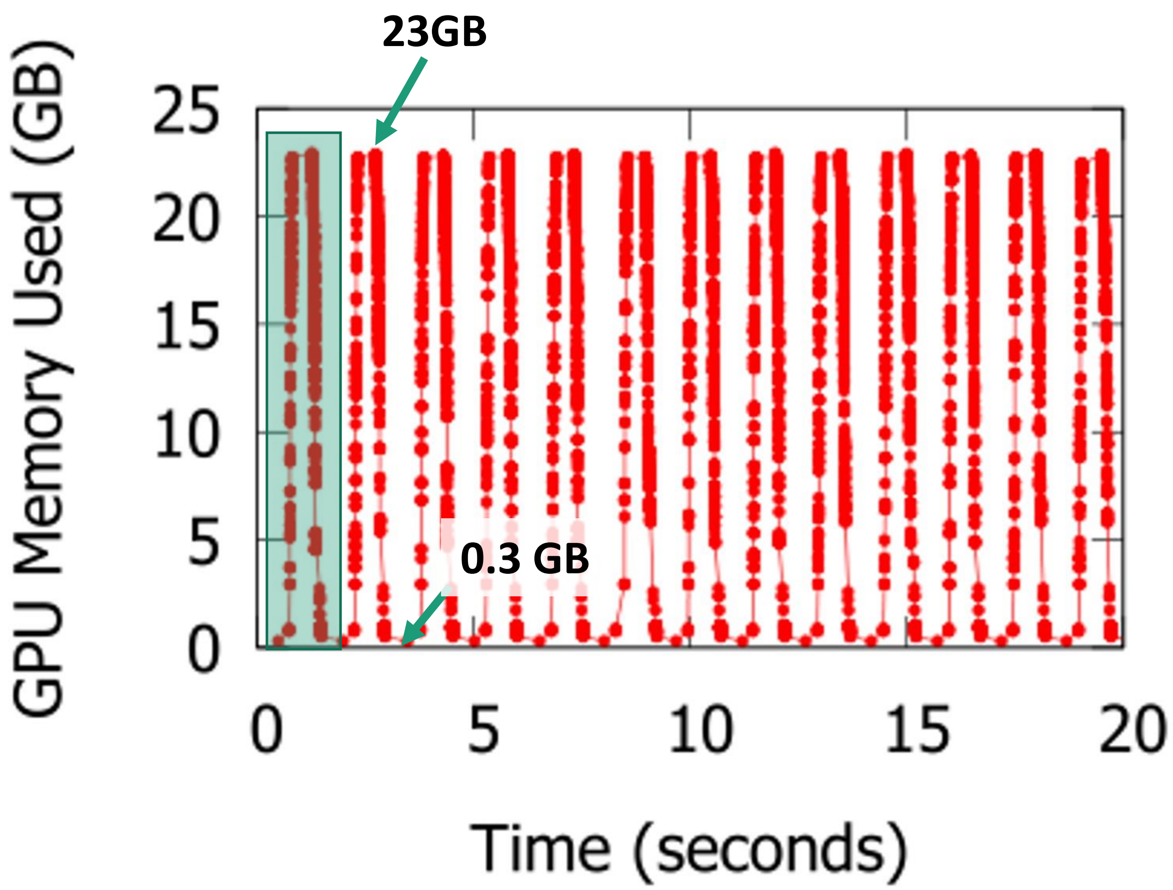
- Treat DLT jobs as generic big-data jobs (e.g. use Yarn, Kubernetes)
- Schedule a job on a GPU exclusively, job holds it until completion
- Problem #2: **Low Efficiency** (Fixed decision at job-placement time)



Need ability to migrate jobs

Sensitivity to locality varies across jobs

Domain knowledge: Intra-job predictability



ResNet50 training on ImageNet data

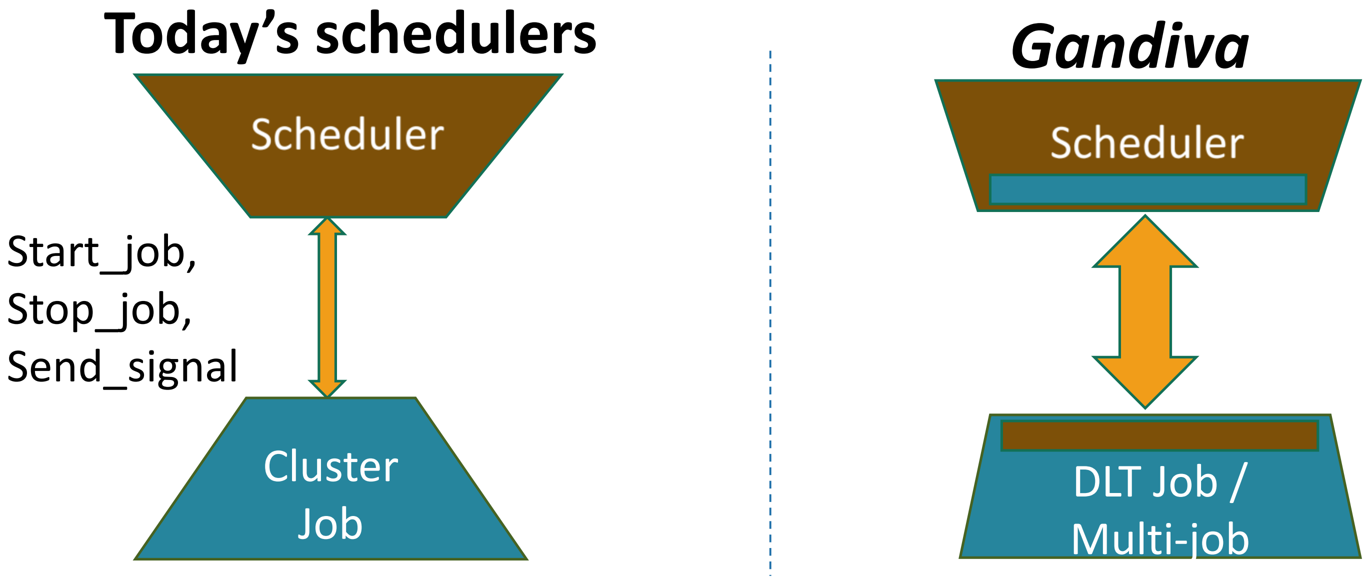
Each spike is a "mini-batch"

Mini-batch times identical

~77x diff. in RAM usage

Time-slicing quantum = Group of minibatches

Gandiva: A domain-specific scheduler for DLT



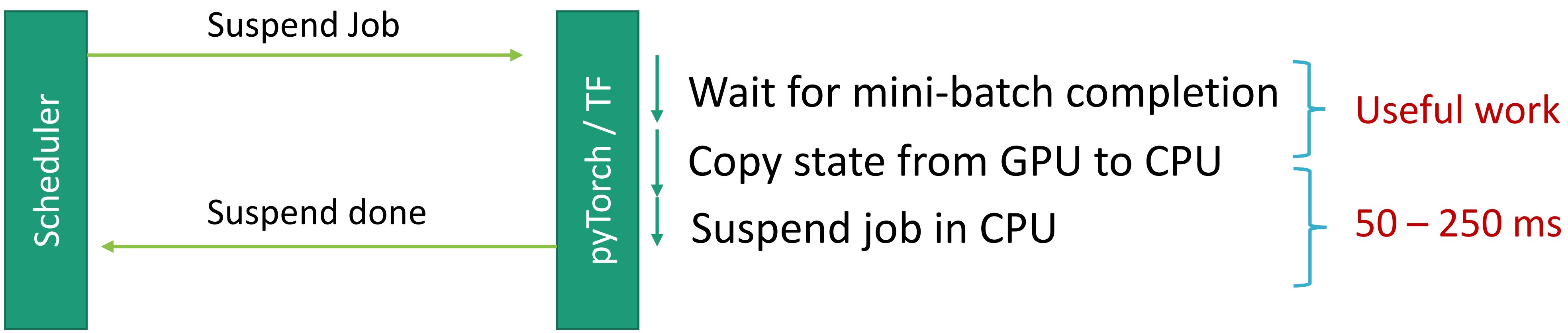
- **<u>Result</u>**: Faster & cheaper execution of DLT workflows
  - <u>Latency</u>: 4.5x lower queueing times, 5-7x faster multi-jobs (AutoML)
  - <u>Efficiency</u>: 26% higher cluster throughput

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



<u>Customization</u>: Align with mini-batch boundary => ~50x cheaper

Migration / Packing

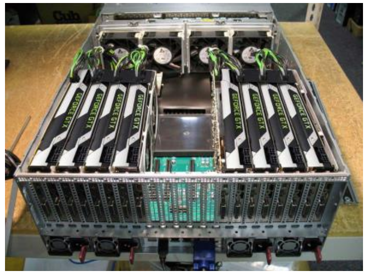
- Move jobs across GPUs to improve efficiency
- Generic distributed process migration is unreliable / slow
  - Customization: Integration with toolkit checkpointing makes it fast/robust

- #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

Job 1

Job 2



~~GPU Util: 50%~~

GPU Util: 80%

Two possibilities:

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

- Solution: Measure useful work directly
  - <u>Customization</u>: Job runtime exports “time-per-minibatch”

- Allows simple “introspection” policy
  - Try migration/packing, measure benefit, revert if negative

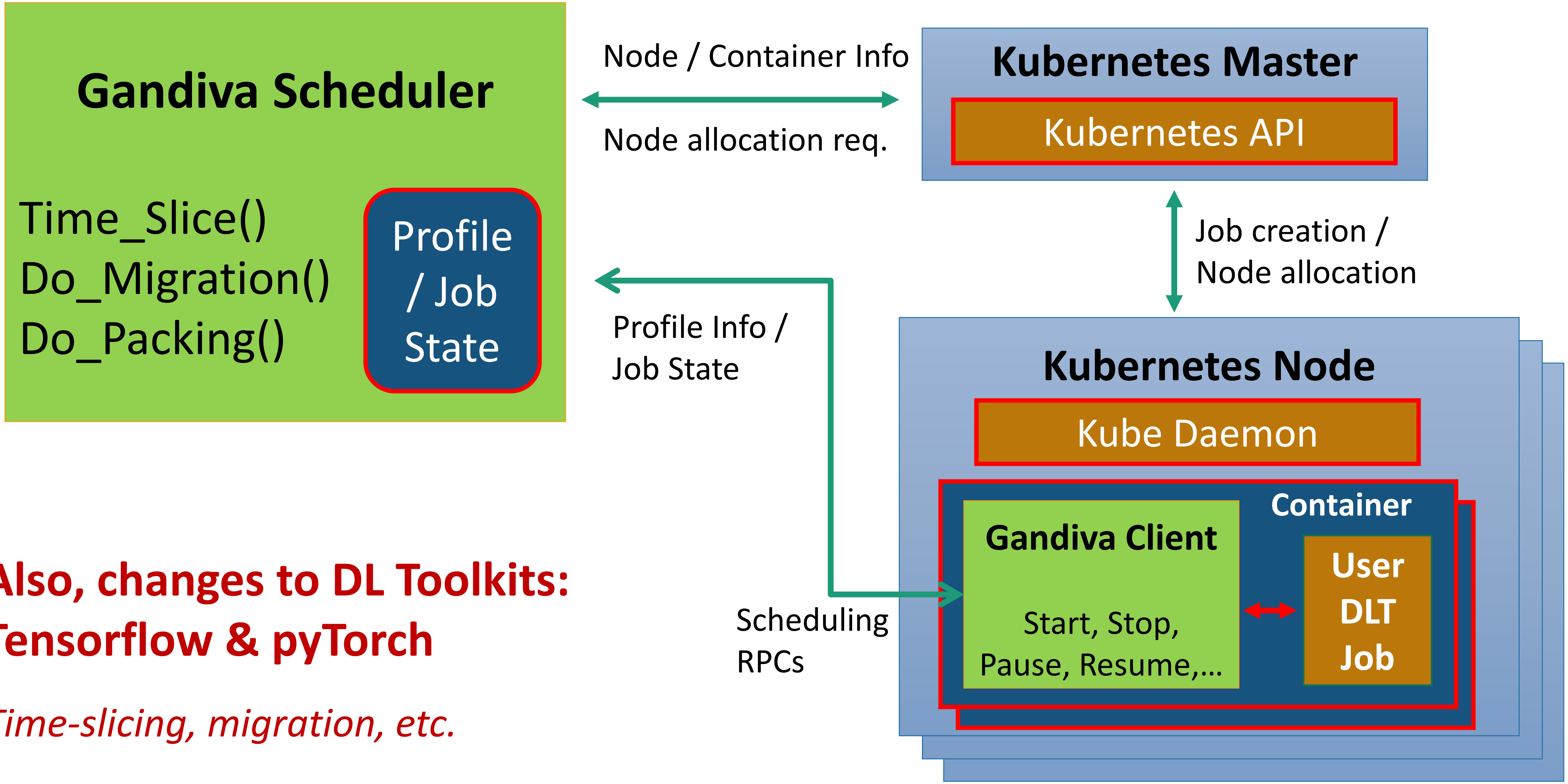
Introspective Scheduling

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

Outline

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

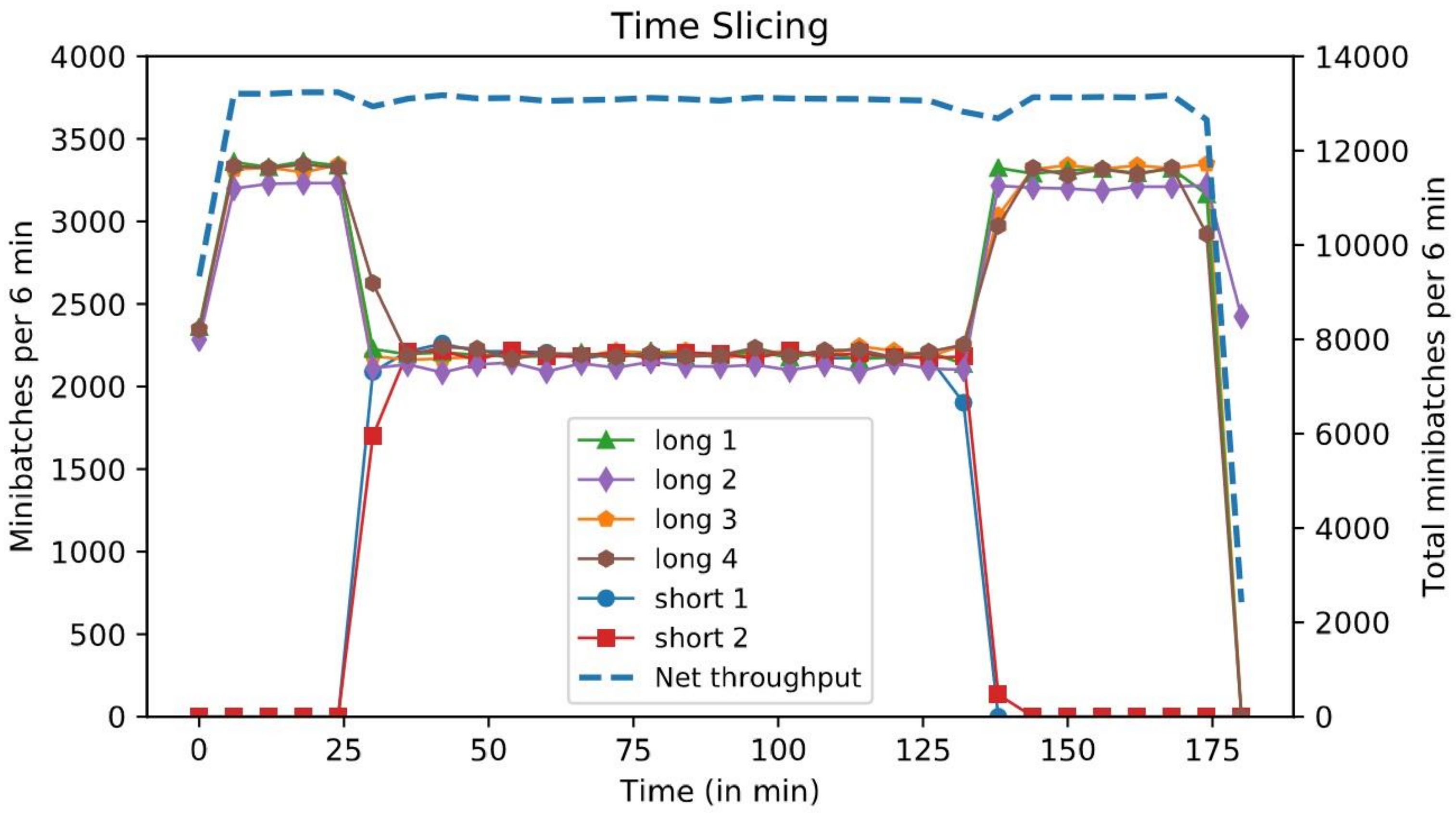
Implementation



Also, changes to DL Toolkits: Tensorflow & pyTorch

Time-slicing, migration, etc.

Microbenchmark: Time-slicing



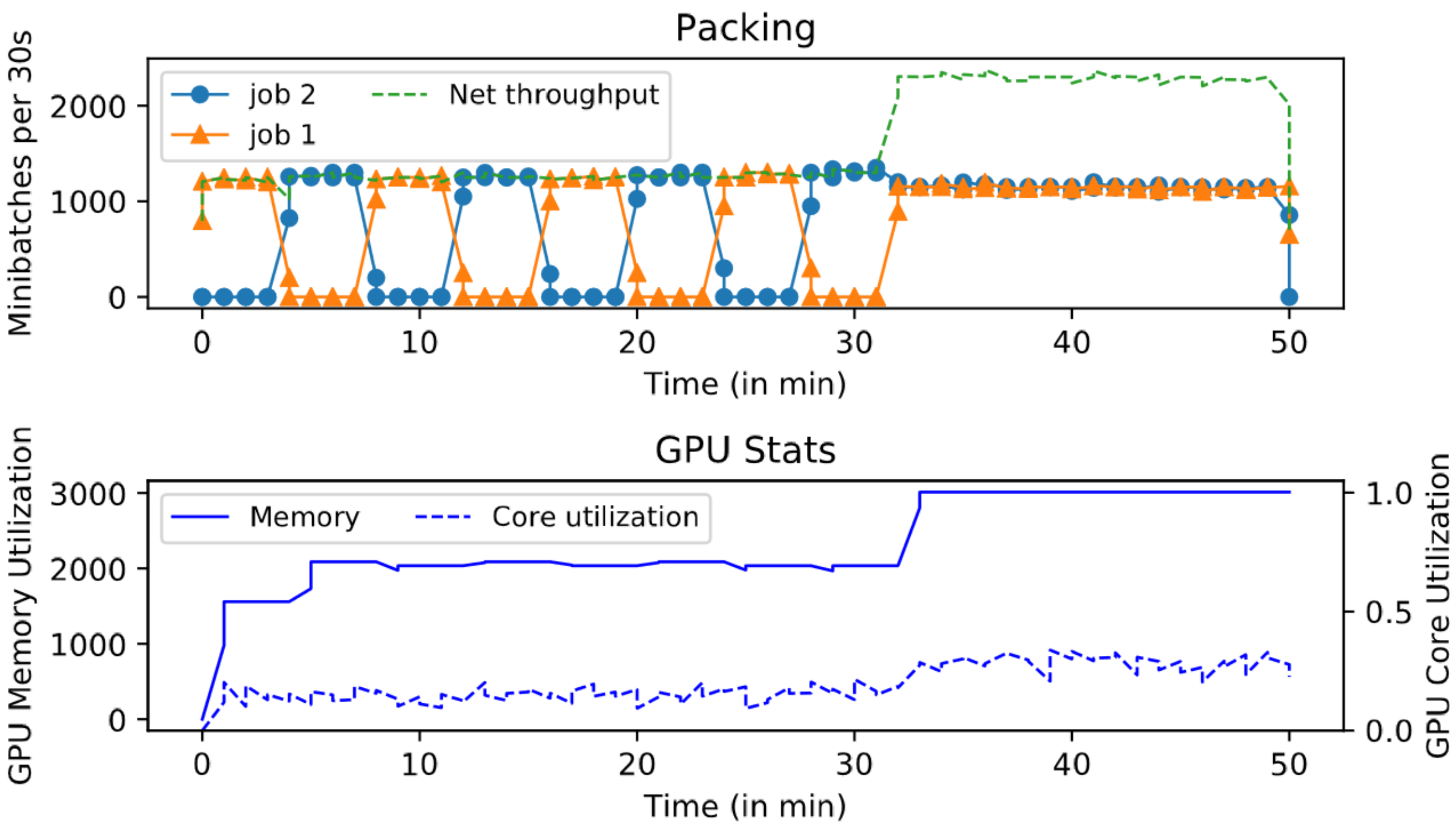
Server 4 P100 GPUs

6 DLT jobs: ResNet50/ImagNet on pyTorch

All jobs get equal time-share during time-slicing

<u>Low overhead:</u>
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

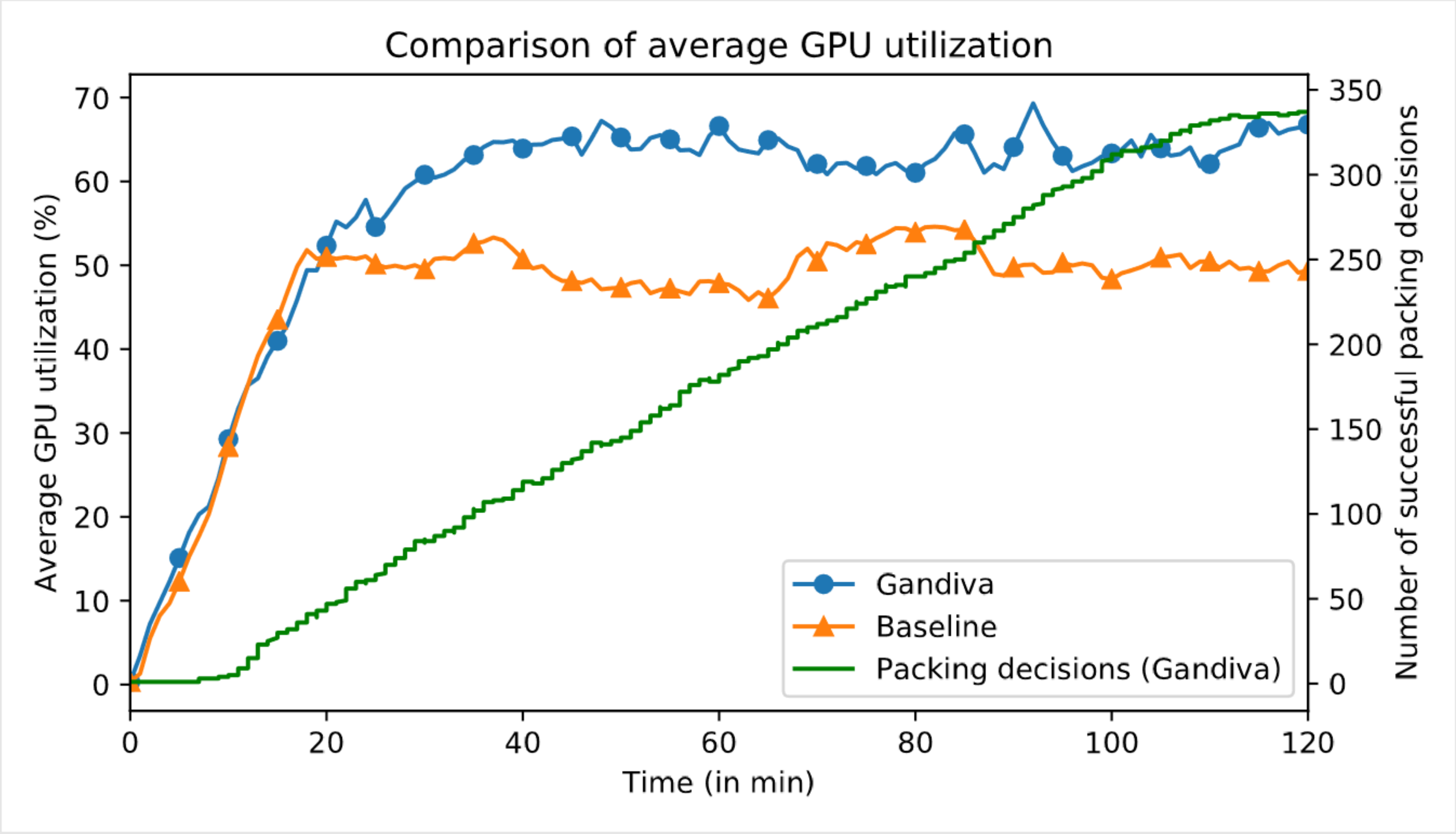
- ResNet-like Model for CIFAR Image dataset; 16 P40 GPUs
- HyperOpt: Predict "more promising" mode based on early feedback

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

	Accuracy: 70%	Accuracy: 80%	Accuracy: 90%
Baseline	134.1	2489.1	5296.7
Gandiva	134.1	543.1	935.4
Speedup	1x	5.25x	5.66x

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