COMS E6998 010

Practical Deep Learning Systems Performance

Lecture 10 11/19/20

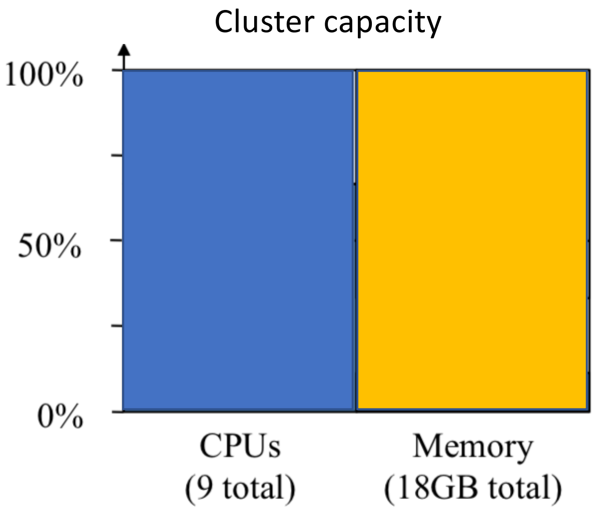
Logistics

- Homework 4 due Nov 22 11:59 PM
- Homework 5 will be posted Nov 22.
- Quiz 4 will be administered in the first week of Dec.

Recall from last lecture

- Deep learning benchmarking
- Tensorflow benchmarks
- DAWNBench
- MLPerf
- Time to Accuracy (TTA) metric: variability, scaling, generalization
- Kaggle, OpenML
- ONNX and ONNX runtime architecture

Resource allocation problem



Job type	Resource demand
A	<1 CPU, 4GB>
B	<3 CPU, 1GB>

How many jobs of each type to run on the cluster ?

- Maximize resource usage
- Be fair to different job types

Dominant Resource Fairness (DRF)

Fairness criteria: Equalize dominant resource share among all the active job types

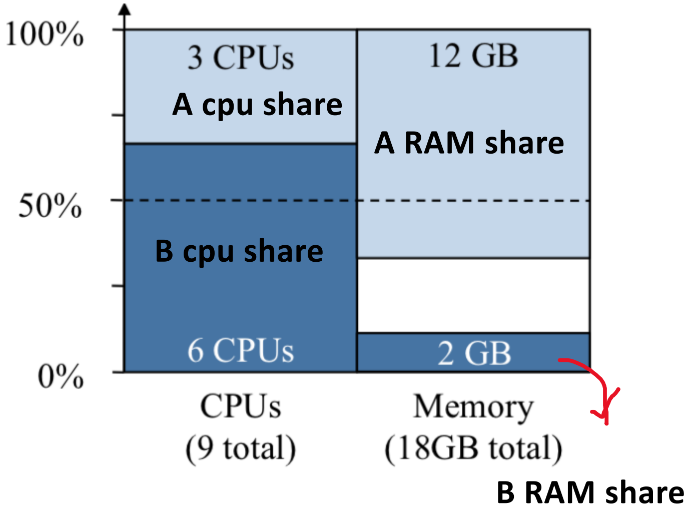
What is the dominant resource for a job type?

- Compute the share of each resource allocated to a job type
- Maximum among all shares of a job type is called that job type's *dominant share*
- Resource corresponding to the dominant share is called the *dominant resource*.

Job type	Resource demand	Number of jobs	Resource share per job	Dominant resource share for all jobs	Dominant resource
A	<1 CPU, 4GB>	x	<1/9, **4/18**>	2x/9	Memory
B	<3 CPU, 1GB>	y	<**3/9**, 1/18>	y/3	CPU

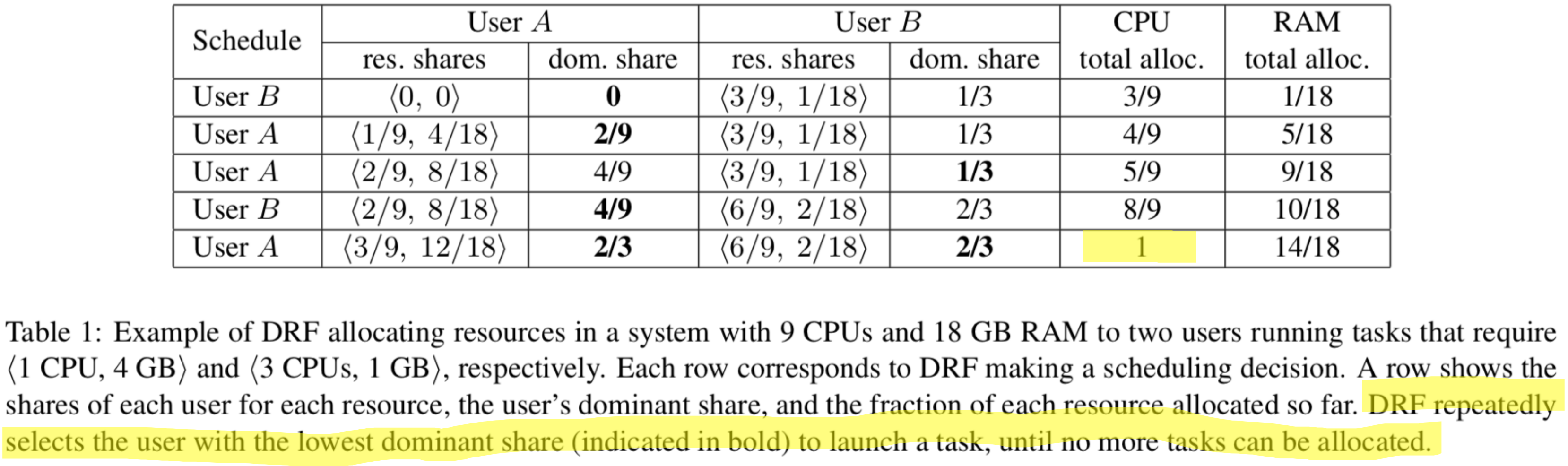
Dominant Resource Fairness (DRF)

$$\begin{array}{rcll} \max\,(x, y) & & & \text{(Maximize allocations)} \\ \text{subject to} & & & \\ x + 3y & \leq & 9 & \text{(CPU constraint)} \\ 4x + y & \leq & 18 & \text{(Memory constraint)} \\ \dfrac{2x}{9} & = & \dfrac{y}{3} & \text{(Equalize dominant shares)} \end{array}$$



Each job-type ends up with the *same* dominant share, *i.e.,* A gets 2/3 of RAM, while user B gets 2/3 of the CPUs.

Dominant Resource Fair (DRF) Scheduling Algorithm



Schedule	User A		User B		CPU	RAM
	res. shares	dom. share	res. shares	dom. share	total alloc.	total alloc.
User B	$\langle 0,\ 0\rangle$	**0**	$\langle 3/9,\ 1/18\rangle$	1/3	3/9	1/18
User A	$\langle 1/9,\ 4/18\rangle$	**2/9**	$\langle 3/9,\ 1/18\rangle$	1/3	4/9	5/18
User A	$\langle 2/9,\ 8/18\rangle$	4/9	$\langle 3/9,\ 1/18\rangle$	**1/3**	5/9	9/18
User B	$\langle 2/9,\ 8/18\rangle$	**4/9**	$\langle 6/9,\ 2/18\rangle$	2/3	8/9	10/18
User A	$\langle 3/9,\ 12/18\rangle$	**2/3**	$\langle 6/9,\ 2/18\rangle$	**2/3**	1	14/18

Table 1: Example of DRF allocating resources in a system with 9 CPUs and 18 GB RAM to two users running tasks that require ⟨1 CPU, 4 GB⟩ and ⟨3 CPUs, 1 GB⟩, respectively. Each row corresponds to DRF making a scheduling decision. A row shows the shares of each user for each resource, the user's dominant share, and the fraction of each resource allocated so far. DRF repeatedly selects the user with the lowest dominant share (indicated in bold) to launch a task, until no more tasks can be allocated.

Optimus: Deep Learning Cluster Scheduler

- A cluster scheduler for deep learning jobs, which
  - Minimizes job training time, and
  - Improves resource usage efficiency in the cluster
- Focus on **data-parallel DL training jobs using the parameter server framework**
- **Approach:**
  - Build resource-performance models for jobs, and
  - Dynamically schedule resources to jobs based on job progress and the cluster load to *minimize average job completion time* and *maximize resource efficiency*
- Scheduling algorithm has two parts
  - Resource allocation
  - Task placement
  - First find resource allocation and then do task placement in the cluster
    - Which node to schedule workers and parameter servers of jobs

Optimus Performance Prediction Models

- Refer to Lecture 8
  - Convergence model
  - Resource-speed model

- Time to completion
  - Combining convergence and resource-speed model

Resource Allocation: Optimization Problem

- Optimization Problem: Minimize average completion time of active jobs

$$\text{minimize} \quad \sum_{j \in J} t_j$$

Total completion time of active jobs

$$\text{subject to:} \quad t_j = \frac{Q_j}{f(p_j, w_j)} \qquad \forall j \in J$$

nonlinear, non-convex

$$\sum_{j \in J} (w_j \cdot O_j^r + p_j \cdot N_j^r) \le C_r \qquad \forall r \in R$$

$$p_j \in Z^+, w_j \in Z^+ \qquad \forall j \in J$$

J: set of current active jobs

t_j: remaining completion time of job j

Q_j: remaining steps/epochs to complete job j

$f(p_j, w_j)$: training speed of job as a function of allocated number of parameter servers p_j and number of workers w_j

R: set of available resource types, e.g., CPU, GPU, memory

O_j^r: amount of type r resource consumed by one worker of job j

N_j^r: amount of type r resource consumed by one parameter server of job j

Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018

Resource Allocation: Marginal Gain

- Reduction in job completion time (JCT) when one PS is added to job j

$$\left(\frac{Q_j}{f(p_j, w_j)} - \frac{Q_j}{f(p_j + 1, w_j)}\right)$$

- Reduction in JCT when one worker is added to job j

$$\left(\frac{Q_j}{f(p_j, w_j)} - \frac{Q_j}{f(p_j, w_j + 1)}\right)$$

Peng et al. Optimus: An Efficient Dynamic Resource Scheduler
for Deep Learning Clusters. Eurosys 2018

Marginal gain in JCT reduction

- A dominant resource is the type of resource that has the maximal share in the overall capacity of the cluster, among all resources used by a worker (parameter server)
- Marginal gain in job completion time reduction per unit dominant resource consumption

$$\begin{aligned}\max\{(\frac{Q_j}{f(p_j, w_j)} - \frac{Q_j}{f(p_j + 1, w_j)})/N_j^D, & \quad \longrightarrow \text{dominant resource for a PS of job j}\\ (\frac{Q_j}{f(p_j, w_j)} - \frac{Q_j}{f(p_j, w_j + 1)})/O_j^{D'}\} & \longrightarrow \text{dominant resource for a worker of job j}\end{aligned}$$

- Marginal gain calculated using prediction model

Marginal Gain based Heuristic for Resource Allocation

- In each scheduling interval
  - First allocate one worker and one parameter server to each active job to avoid starvation
  - Sort all jobs in order of their computed marginal gains
  - Iteratively select the job with the largest marginal gain and add one worker or parameter server to the job, depending on whether adding a worker or adding a parameter server brings larger marginal gain
  - Marginal gains of the jobs are updated when their resource allocation changes. The procedure repeats until some resource in the cluster is used up, or marginal gains of all jobs become non-positive.

- How to take care of inaccuracies in prediction model during early stages of a job ?

Task Placement

- Placement of workers and PSs on different servers in the cluster
- Reduce processing time at workers/PSs improve by reducing the time spent on parameters/gradients exchange among workers and PSs

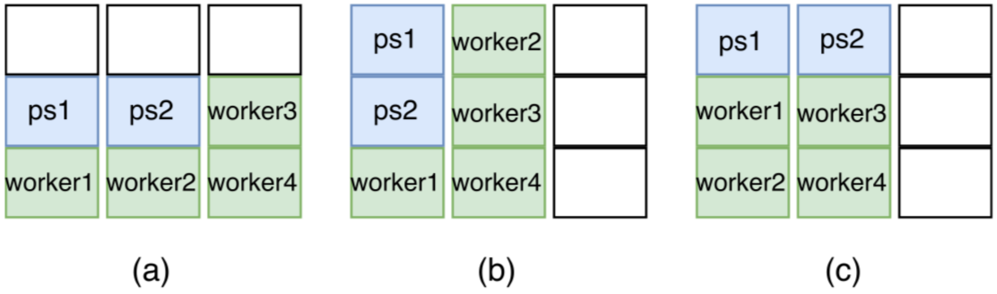


Figure 10: An example of worker/parameter server placement: (c) is the best

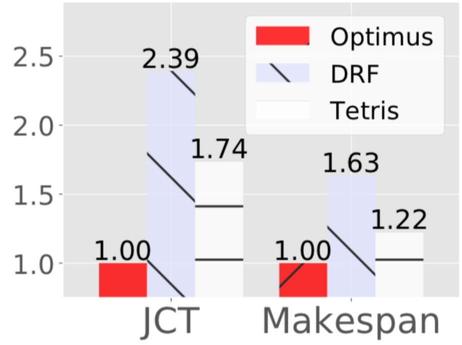
Component	Units of data to transfer		
	(a)	(b)	(c)
ps1	**3**	**3**	**2**
ps2	**3**	**3**	**2**
worker1	1		1
worker2	1	2	1
worker3	2	2	1
worker4	2	2	1

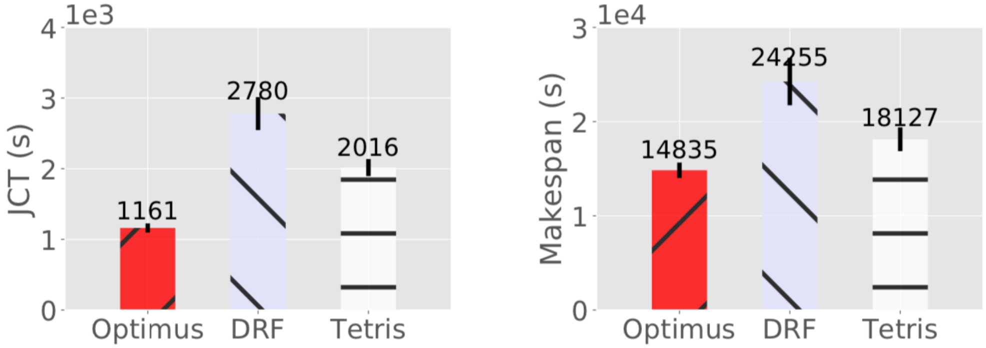
Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018

Task Placement Approach

- *Approach*: Use **the smallest number of servers** to host the job, such that the **same number of parameter servers and the same number of workers** are deployed on each of these servers
- Optimality established for synchronous training on a cluster of homogeneous servers
- Principle:
  - colocating workers and parameter servers can reduce cross-server data transfers
  - packing the same number of workers/parameter servers of a job on each server can minimize the maximal data transfer time in each step of synchronous training

Job Completion Time (JCT) and Makespan



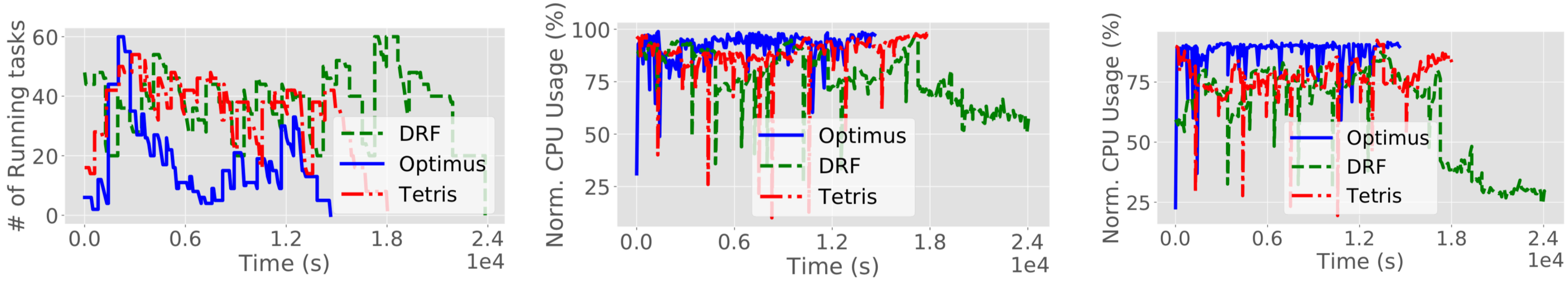


(a) JCT **(b) Makespan**

- Makespan: Total time elapsed from the arrival of the first job to the completion of all jobs
  - Minimizing makespan is equivalent to maximizing resource efficiency
- Reduction in JCT and makespan by 2.39x and 1.63x respectively
- Optimus, DRF and Tetris use 4.1, 6.7 and 5.0 hours to finish all jobs
- DRF applies load balancing when doing task placement
- Tetris packs jobs to servers to minimize resource fragmentation

Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018

Resource utilization

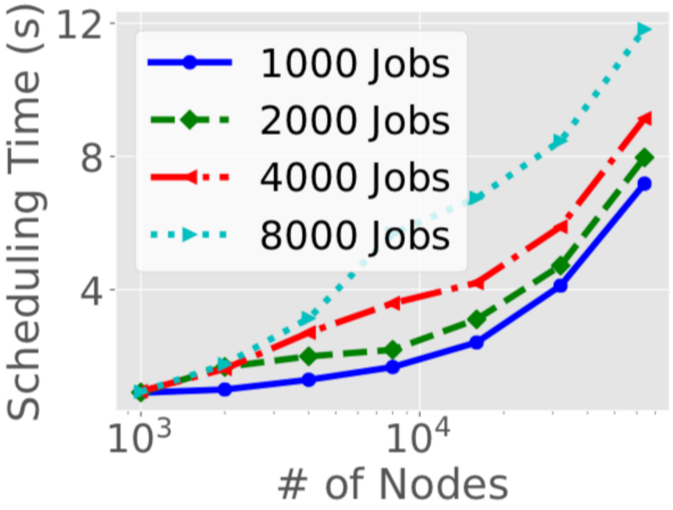


(a) Number of running tasks **(b) CPU usage on parameter servers** **(c) CPU usage on workers**

- Workload: Job arrival happens randomly between [0,12000] seconds
- DRF is work-conserving and allocates as many resources to a job as possible
- Optimus allocates resources to a job only if there is an improvement in speed
- DRF runs more tasks compared to Optimus
  - CPU utilization is higher with Optimus for both workers and PSs
  - Makespan is smaller with Optimus

Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018

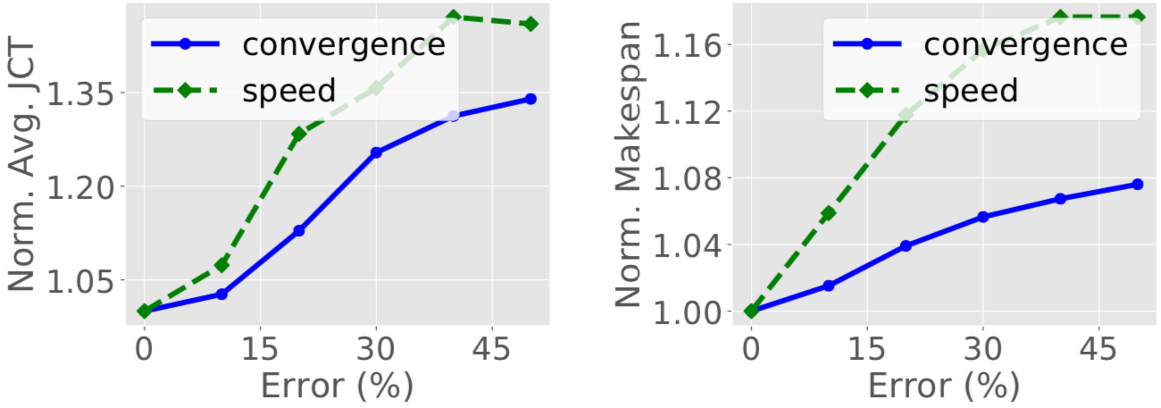
Optimus Scalability to Large Clusters



- Can schedule 4,000 jobs (about 100,000 tasks) within 5 seconds on a cluster of 16,000 nodes
- Comparable to the performance of Kubernetes' default scheduler, i.e., 150,000 tasks in 5,000 nodes within 5 seconds
- Optimus makes scheduling decisions at each scheduling interval (~10 mins)
- Overall resource adjustment overhead is 2.54% of the makespan

Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018

Optimus Scheduler: Sensitivity to Prediction Error

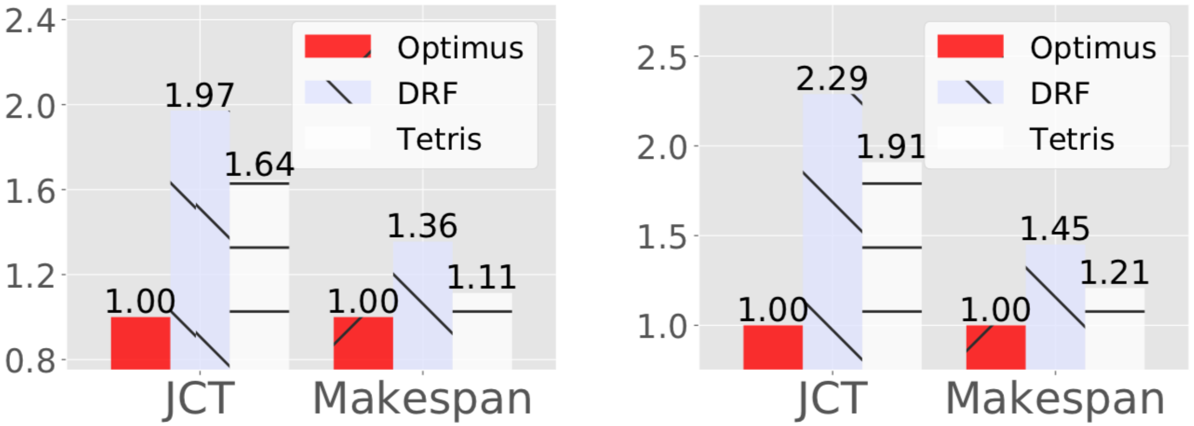


(a) Average completion time **(b) Makespan**

- Optimus scheduler performance is more sensitive to the error in speed estimation compared to the error in convergence estimation
- Downgrade the priority of a job a bit when it is at the beginning state (i.e., larger prediction errors) by multiplying its marginal gain by a factor (e.g., 0.95)
- Smaller marginal gain of a job means less resources allocation, mitigating the influence of large prediction errors at the start of training

Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018

Optimus Scheduler under Sync and Async

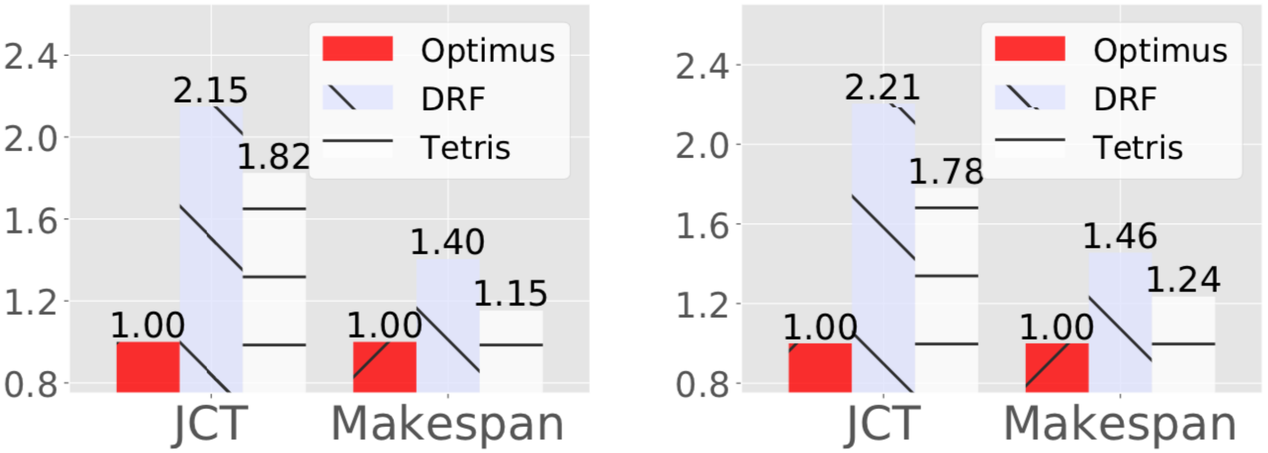


(a) Async **(b) Sync**

- Performance gains with synchronous are higher than with asynchronous
- Both convergence estimation error and speed estimation error are small with synchronous

Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018

Optimus Scheduler: Sensitivity to Workload



(a) Poisson **(b) Google cluster trace**

- Google cluster job arrival trace over a 7 hr long period

Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018

Optimus: Where is the gain coming from ?

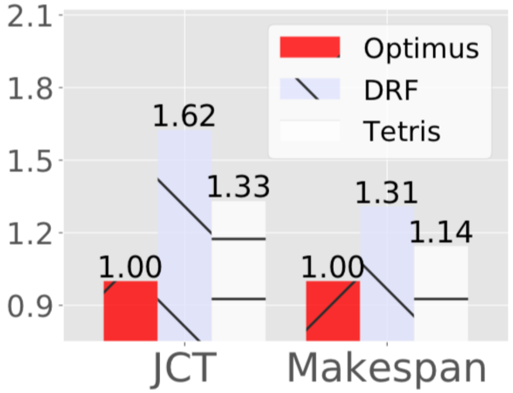


Figure 18: Effectiveness of resource allocation algorithm

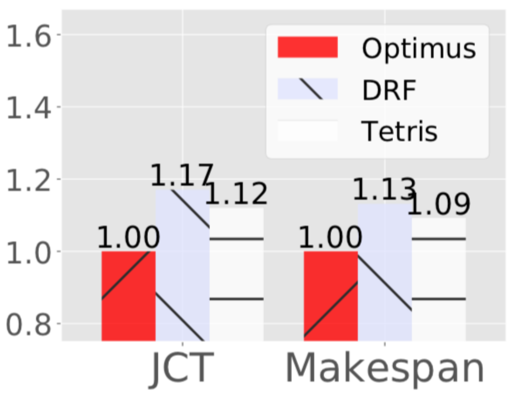


Figure 19: Effectiveness of task placement algorithm

Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018

Problem with current scheduling of Deep Learning Training (DLT) jobs on clusters

- Cloud operators and large companies that manage clusters of tens of thousands of GPUs rely on cluster schedulers to ensure efficient utilization of the GPUs.
- Use a traditional cluster scheduler, such as Kubernetes or YARN treating a DL job simply as yet another big-data job
- DLT jobs are assigned a fixed set of GPUs at startup
  - Job holds exclusive access to its GPUs until completion
  - Head-of-line blocking, preventing early feedback and resulting in high queuing times for incoming jobs
  - Low GPU utilization
- Existing schedulers treat DLT job as a black box

DLT Job Features

- Short jobs vs long jobs on a DLT cluster
- *Feedback-driven exploration*
  - Hyperparameter search; manual or automated
  - Do not need to run jobs to completion

with identified right hyperparameters

  - Require early indication of performance
- *Head-of-line-blocking*, as long-running jobs hold exclusive access to the GPUs until completion, while multi- jobs depending on early feedback wait in queue
- DLT jobs are heterogeneous targeting diverse domains
  - Jobs widely differ in terms of memory usage, GPU core utilization, sensitivity to interconnect bandwidth, and/or interference from other jobs

DL Jobs: Sensitivity to Locality

Single node setting

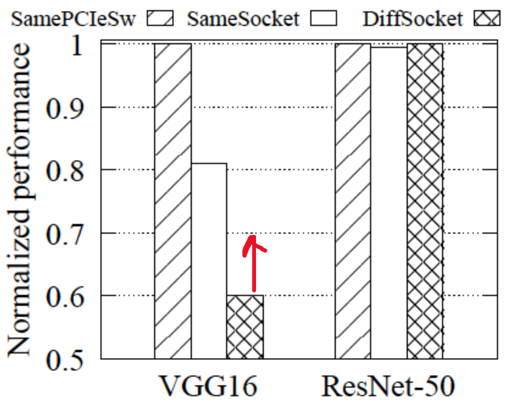


Figure 1: Intra-server locality.

Distributed setting

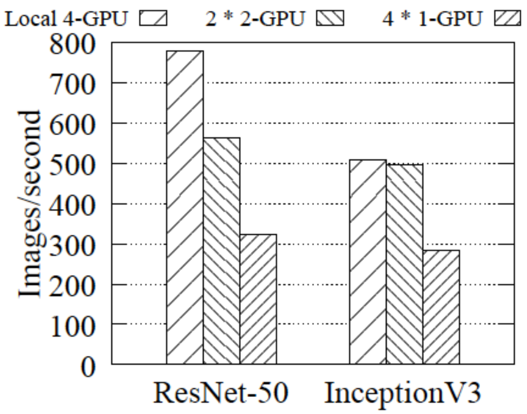


Figure 2: Inter-server locality.

- DiffSocket: GPUs on different CPU sockets
- SameSocket: GPUs in the same CPU socket, but on different PCIe switches
- SamePCIeSw: GPUs on the same PCIe switch

- Different DLT jobs exhibit different levels of sensitivity to inter-GPU affinity. Even for GPUs on the same machine, we observe different levels of inter-GPU affinity due to asymmetric architecture
- Sensitivity is model dependent; VGG16 is a larger neural model than ResNet-50
  - Model synchronization in each mini-batch incurs a higher com- munication load on the underlying PCIe bus.
- DLT scheduler need to take into account a job's sensitivity to locality when allocating GPUs

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

DL Jobs: Sensitivity to Interference

Intra-server interference

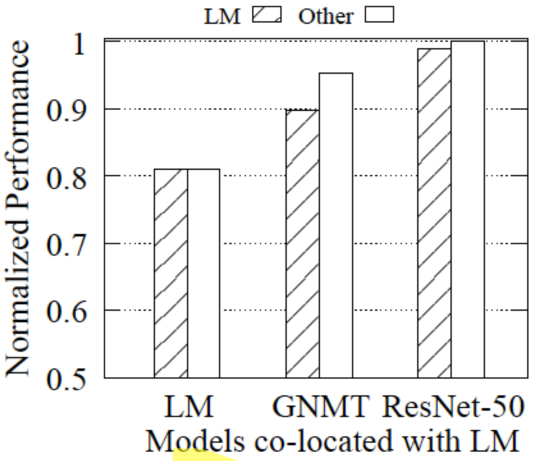


Figure 3: 1-GPU interference.

Inter-server interference

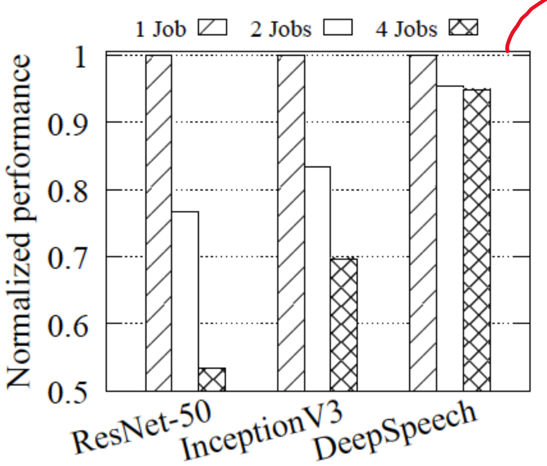


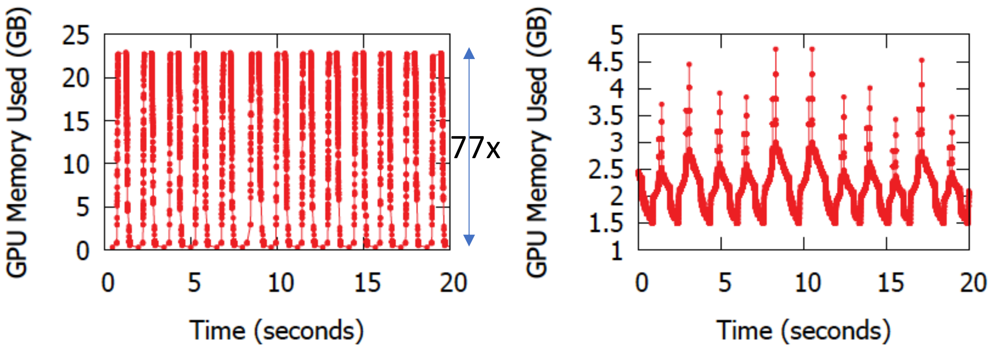
Figure 4: NIC interference.

- 2 4-GPU servers connected by 40GB InfiniBand network
- 2-GPU jobs with each GPU on a different server

- Jobs interfere caused by resource contention due the presence of multiple jobs
- Different DLT jobs exhibit different degrees of interference
- Interference exists both for single-GPU and multi-GPU jobs
- When running multiple 2-GPU jobs, where each GPU is placed on different server, ResNet-50 shows up to 47% slowdown, InceptionV3 shows 30% slow-down

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

DL Jobs: Intra Job Predictability



(a) ResNet50/Imagenet

(b) GNMT/WMT'14 En-De

Figure 5: GPU memory usage during training.

- Gradient descent algorithm performing many mini-batch iterations
- Cyclic pattern in GPU memory used
- Each cycle corresponds to one mini-batch processing
- Leveraging predictability
  - Define micro-tasks (collection of cycles) as scheduling units
  - Very efficient suspend/resume and migration by performing them at end of cycle when memory usage is minimum
- Mini-batch progress rate can be profiled and used as proxy to evaluate the effectiveness of applying performance optimization mechanisms

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Scheduler

- **Goals**
    - Early feedback
        - Gandiva supports over-subscription by allocating GPUs to a new job immediately and using the suspend-resume mechanism to provide early results
    - Cluster efficiency
        - Through a continuous optimization process that uses profiling and a greedy heuristic that takes advantage of mechanisms such as packing, migration, and grow-shrink
- Modes of operations
    - Reactive
    - Introspective
- Scheduling framework to exploit the unique characteristics of the deep learning workload

Gandiva Mechanisms

- **3 ways to remove exclusivity and fixed assignment of GPUs** to DLT jobs
  - **Time-sharing GPUs:** During overload, instead of waiting for current jobs to depart, *Gandiva* allows incoming jobs to **time-share GPUs** with existing jobs. This is enabled using a custom **suspend-resume mechanism** tailored for DLT jobs along with selective packing.
  - **Job migration:** *Gandiva* supports **efficient migration of DLT jobs** from one set of GPUs to another. Migration allows time-sliced jobs to migrate to other (recently vacated) GPUs or for defragmentation of the cluster so that incoming jobs are assigned GPUs with good locality.
  - **GPU grow-shrink:** *Gandiva* supports a **GPU grow-shrink mechanism** so that idle GPUs can be used opportunistically. In order to support these mechanisms efficiently and enable effective resource management, *Gandiva* introspects DLT jobs by continuously profiling their resource usage and estimating their performance.

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Mechanisms

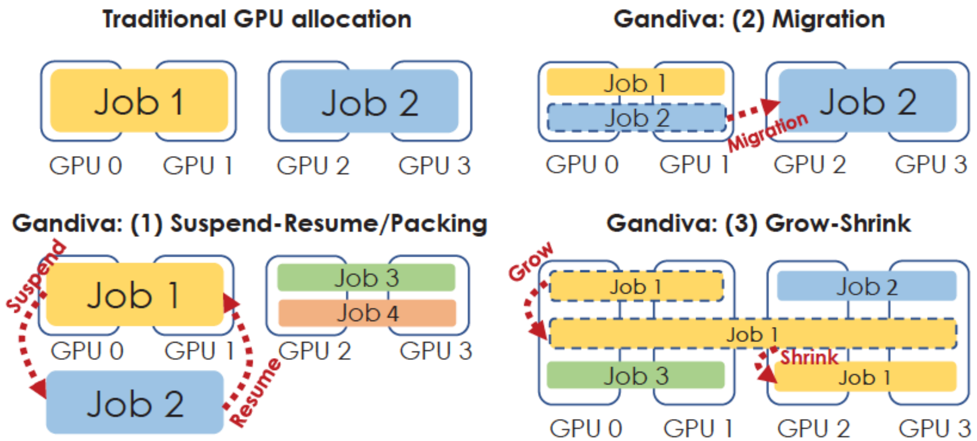


Figure 6: GPU usage options in Gandiva.

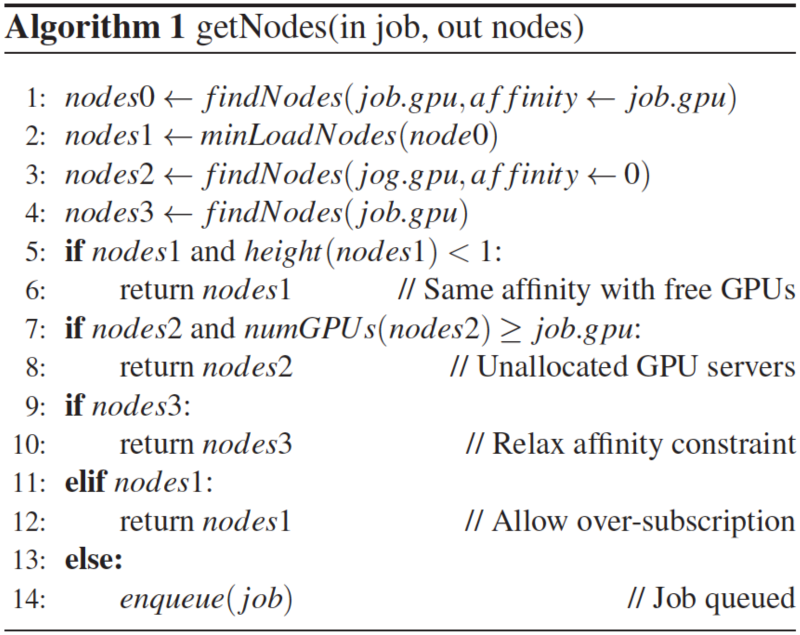
adds custom support for GPU time-slicing.

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Scheduler: Reactive Mode

- Designed to take care of events such as job arrivals, departures, and machine failures
- Conventional schedulers operate in this mode
- Gandiva allows over-subscription
  - When a server is oversubscribed, weighted round-robin scheduling to give each job its fair time-share

Node Allocation Policy in Gandiva



Algorithm 1 getNodes(in job, out nodes)

```
$nodes0 \leftarrow findNodes(job.gpu, affinity \leftarrow job.gpu)$
$nodes1 \leftarrow minLoadNodes(node0)$
$nodes2 \leftarrow findNodes(jog.gpu, affinity \leftarrow 0)$
$nodes3 \leftarrow findNodes(job.gpu)$
if $nodes1$ and $height(nodes1) < 1$:
    return $nodes1$                    // Same affinity with free GPUs
if $nodes2$ and $numGPUs(nodes2) \geq job.gpu$:
    return $nodes2$                    // Unallocated GPU servers
if $nodes3$:
    return $nodes3$                    // Relax affinity constraint
elif $nodes1$:
    return $nodes1$                    // Allow over-subscription
else:
    $enqueue(job)$                     // Job queued
```

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Scheduling Example

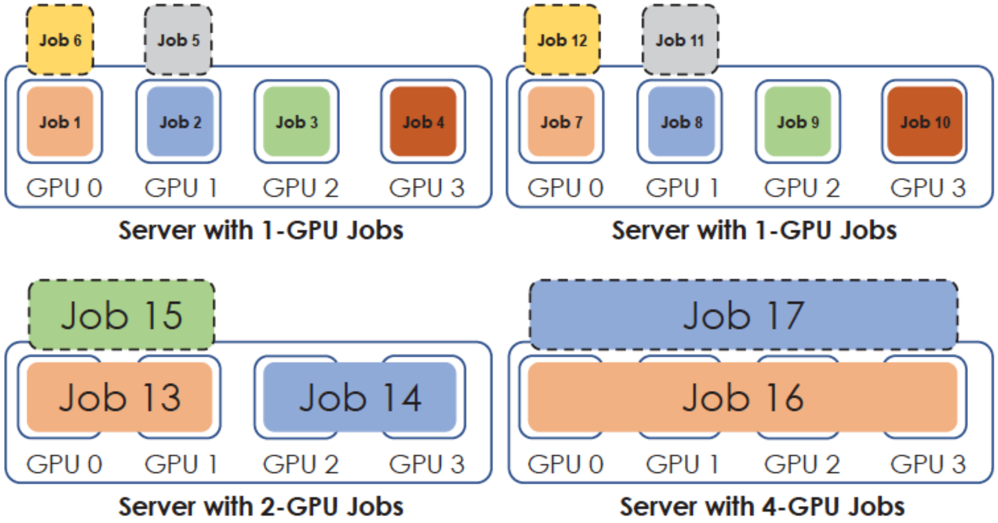


Figure 7: Scheduling example in a 16-GPU Cluster.

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Scheduler: Introspective Mode

- Gandiva continuously monitors and optimizes placement of jobs to GPUs in the cluster to improve the overall utilization and the completion time of DLT jobs
- Introspects DLT jobs to estimate their rate of progress
- Introspection exploits the regular pattern exhibited by DLT jobs and uses the periodicity to estimate their progress rate.
- Techniques in introspective mode:
  - Packing: running more than one job on a GPU
  - Migration
  - Time Slicing
    - Considered only during overload.
    - Run two or more jobs simultaneously on a GPU to increase efficiency
- If a technique is not found to be effective (by comparing mini_batch_time before and after) Gandiva can decide to continue with the technique or not

Job Migration

Co-locate jobs after a departure

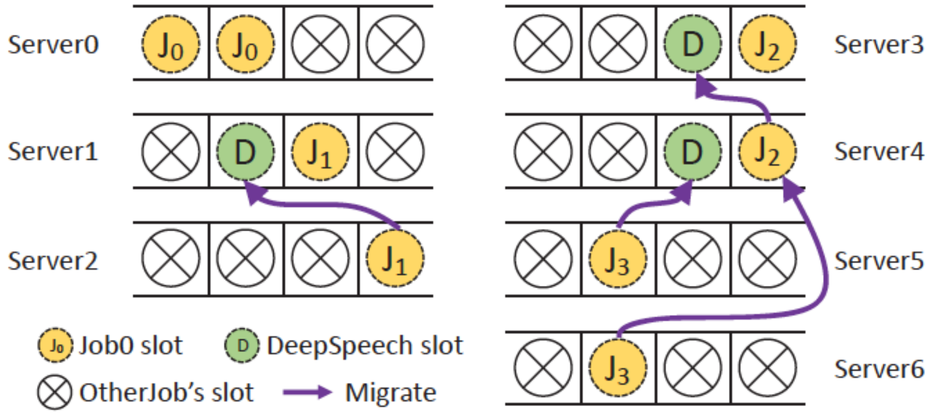


Figure 8: Job migration in a shared cluster.

Migration helpful in
i) Moving time-sliced jobs to vacated GPUs anywhere in the cluster;
ii) Migrating interfering jobs away from each other;
iii) Defragmentation of the cluster so that incoming jobs get GPUs with good locality.

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Performance: Time Slicing

Each of the long job
share of GPU time drop to 4/6

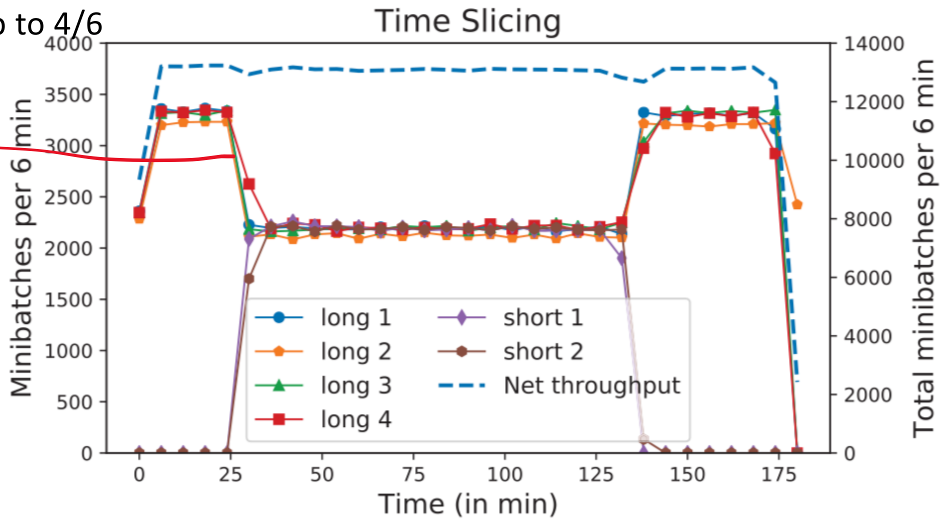
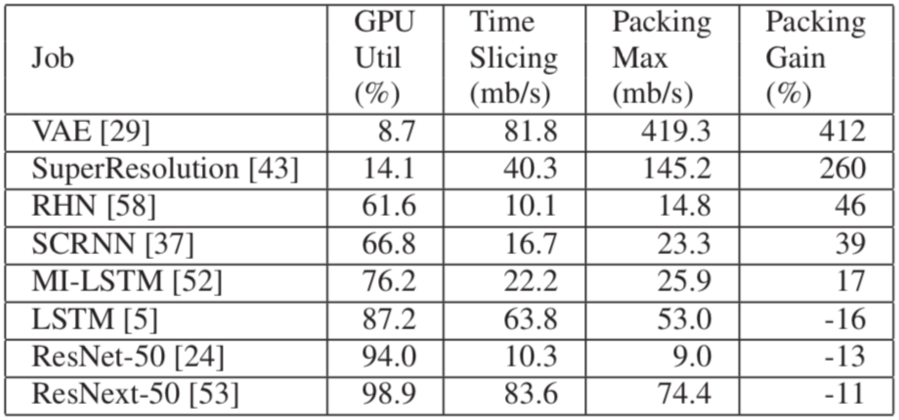


Figure 9: Time slicing six 1-GPU jobs on 4 GPUs.

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Performance: Packing



Job	GPU Util (%)	Time Slicing (mb/s)	Packing Max (mb/s)	Packing Gain (%)
VAE [29]	8.7	81.8	419.3	412
SuperResolution [43]	14.1	40.3	145.2	260
RHN [58]	61.6	10.1	14.8	46
SCRNN [37]	66.8	16.7	23.3	39
MI-LSTM [52]	76.2	22.2	25.9	17
LSTM [5]	87.2	63.8	53.0	-16
ResNet-50 [24]	94.0	10.3	9.0	-13
ResNext-50 [53]	98.9	83.6	74.4	-11

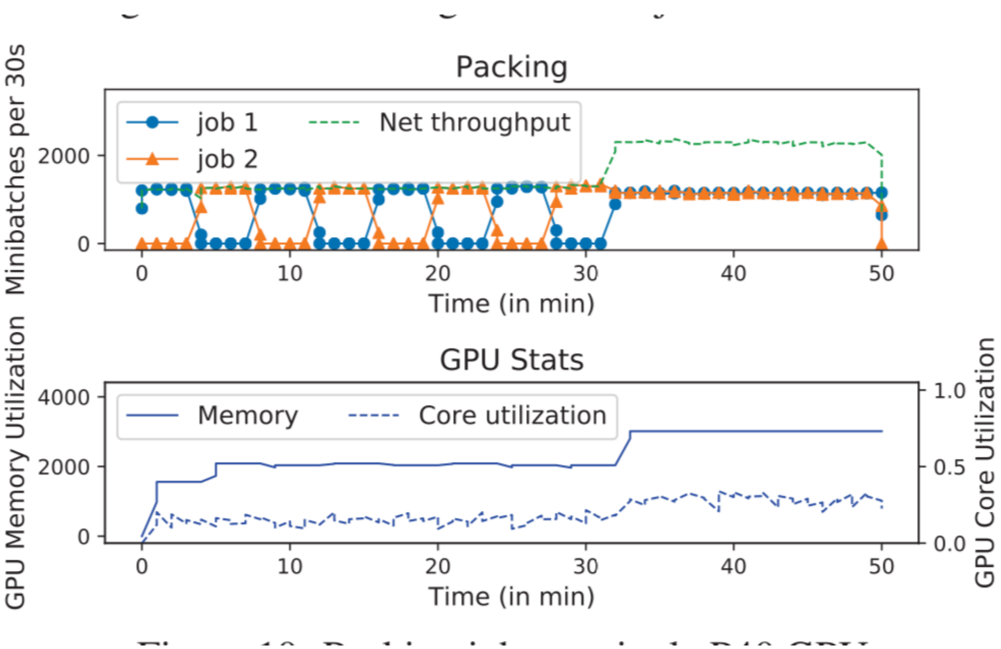
Table 1: Packing multiple jobs on P40 (mb/s = minibatches/s).

Gandiva adopts a profiling-based approach to packing.

If the memory requirements of the packing jobs combined
are higher than GPU memory, the overhead of “paging”
from CPU memory is significantly high that packing is not effective.

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Performance: Packing



Jobs are initially being time- sliced on the same P40 GPU. After some time, the scheduler concludes that their memory and GPU core utilization is small enough that packing them is feasible and schedules them together on the GPU. The scheduler continues to profile their performance. Because their aggregate performance improves, packing is retained; otherwise (not shown), packing is undone and the jobs continue to use time-slicing.

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Performance: Grow-Shrink

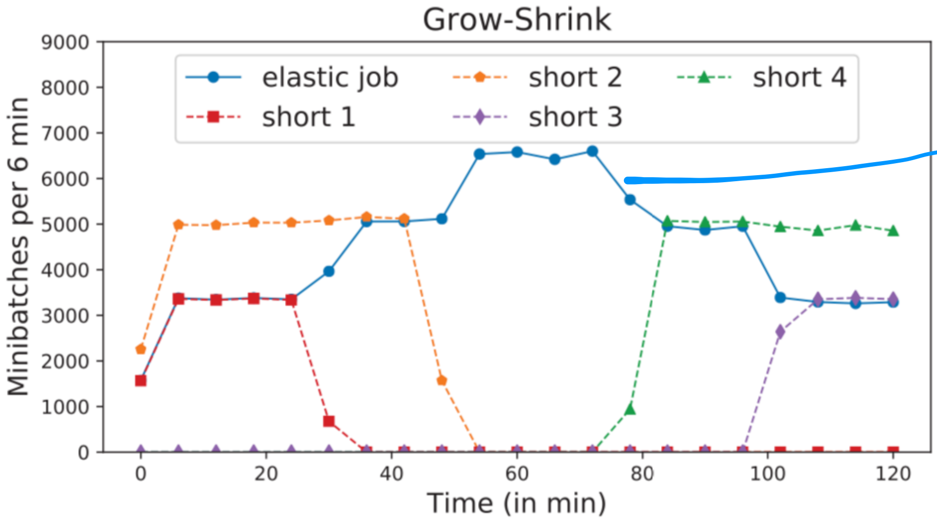


Figure 11: Grow from 1 to 4 GPUs, Shrink to 1-GPU.

Job shrinks and grows

4-P100 server

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Performance: Migration

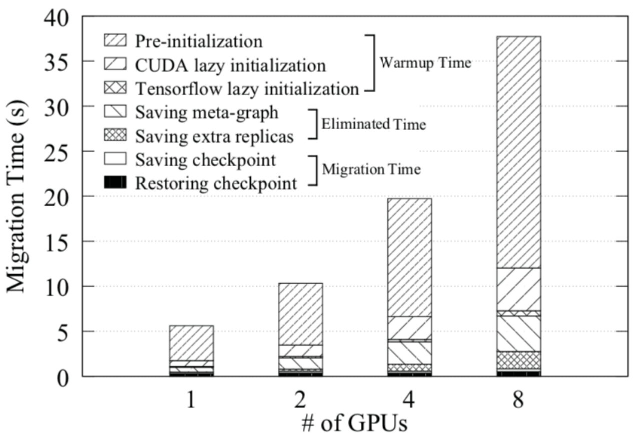


Figure 12: The breakdown of TF migration overheac

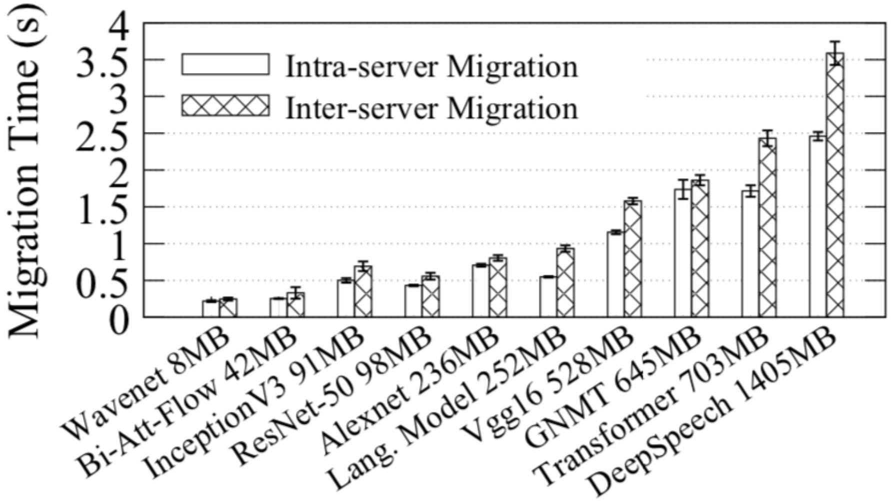


Figure 13: Migration time of real workloads.

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Gandiva Performance: Cluster GPU Utilization

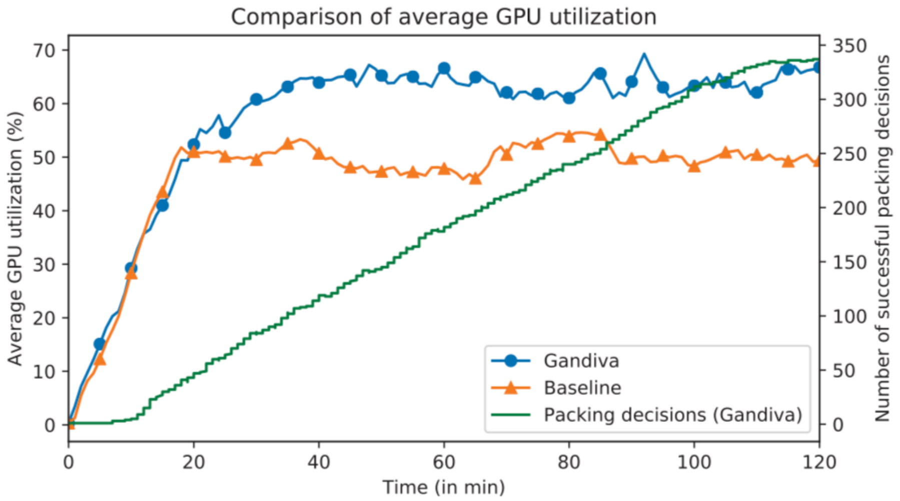


Figure 15: Cluster GPU utilization.

Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018

Seminar Reading List

- Ghodsi et al. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. Usenix NSDI 2011
- **DL Cluster Scheduling**
  - Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning**.** Usenix OSDI 2018
  - Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Eurosys 2018