Tiresias: A GPU Cluster Manager for Distributed Deep Learning

Juncheng Gu, et al.

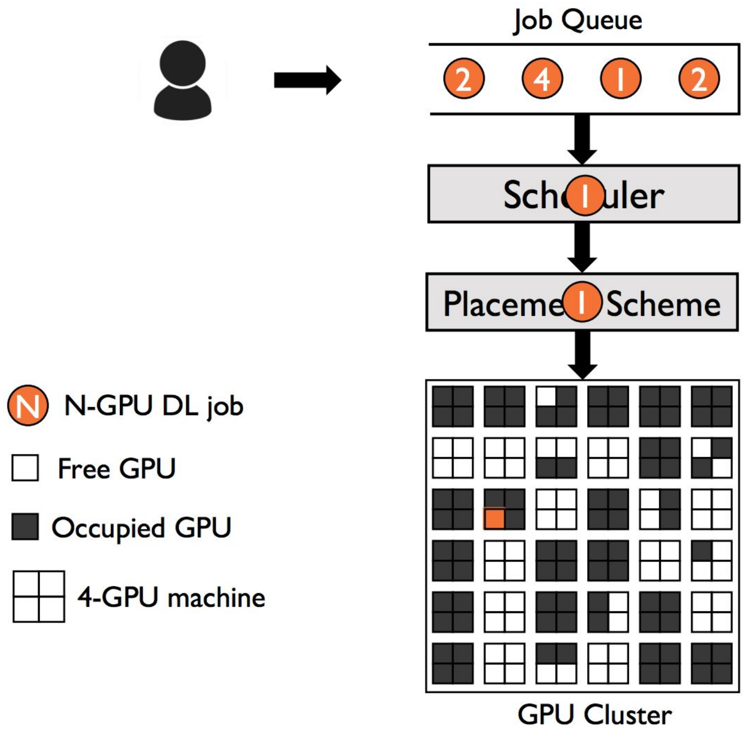
Presenter: Ruiyang Zhu

Tiresias: A GPU Cluster Manager for Distributed Deep Learning

Deep Learning (DL) is popular

- DL training jobs require GPU resources
- GPU clusters are used for training of different jobs
- But how to efficiently manage the GPU resources for DL jobs is a open problem

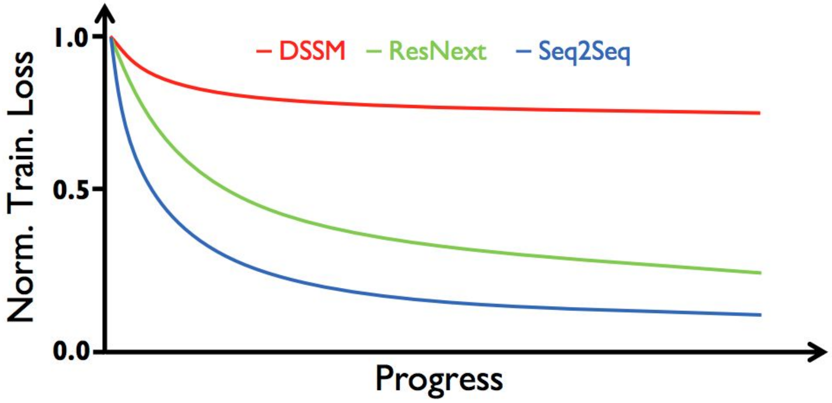
Objectives for DDL training scheduler

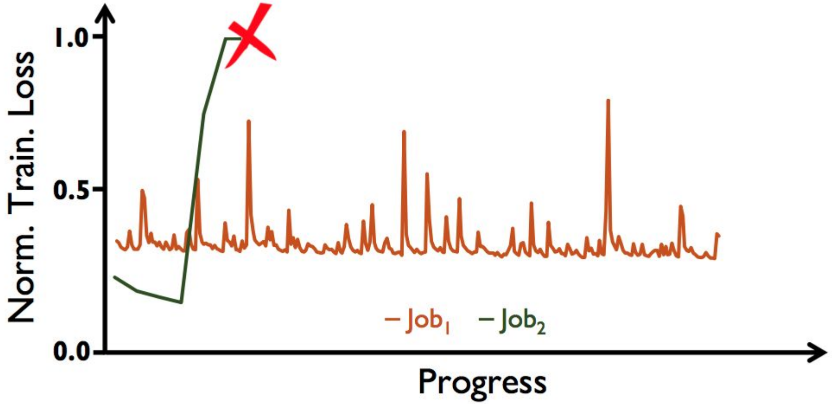


- Minimize Job Completion Time (JCT)
- Achieve high resource utilization
- Fairness among jobs

Challenge I: Unpredictable job duration

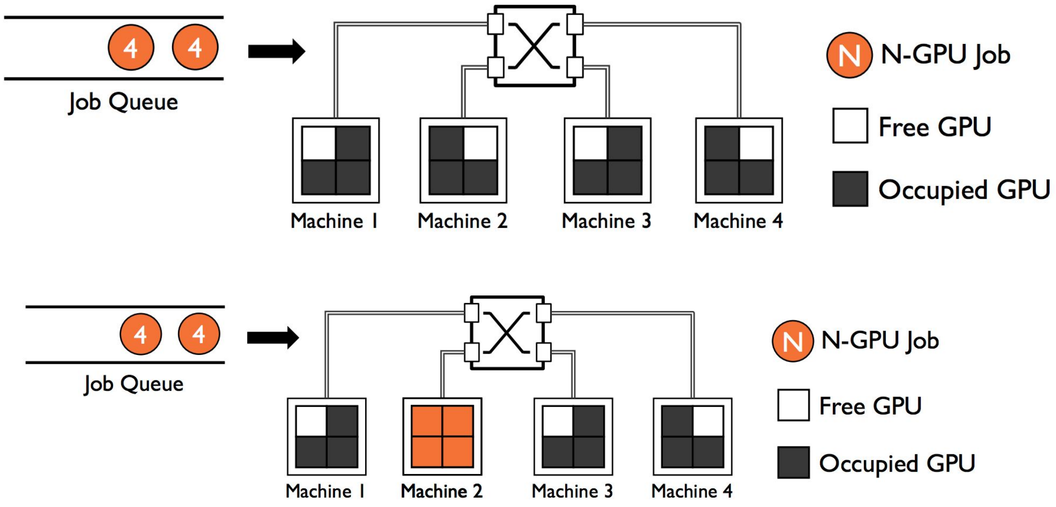
- Unknown execution time of DL training jobs
  - If known, Shortest-remaining-time/service-first (SRTF/SRSF) will be good
- Hard to predict training time (job execution time)
  - Many jobs are part of **trial-and-error process**





Challenge II: Over-Aggressive Job Consolidation

- Existing cluster manager tries to minimize distributing jobs to multiple servers (avoid network overhead)



Previous Solutions

	I: Unpredictable job duration	II: Over-Aggressive Job Consolidation
Optimus [1]	None	None
YARN-CS	FIFO	None
Gandiva [2]	Time-sharing	Trial-and-error

[1] Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters, EuroSys 18

[2] Gandiva: Introspective Cluster Scheduling for Deep Learning, OSDI 18

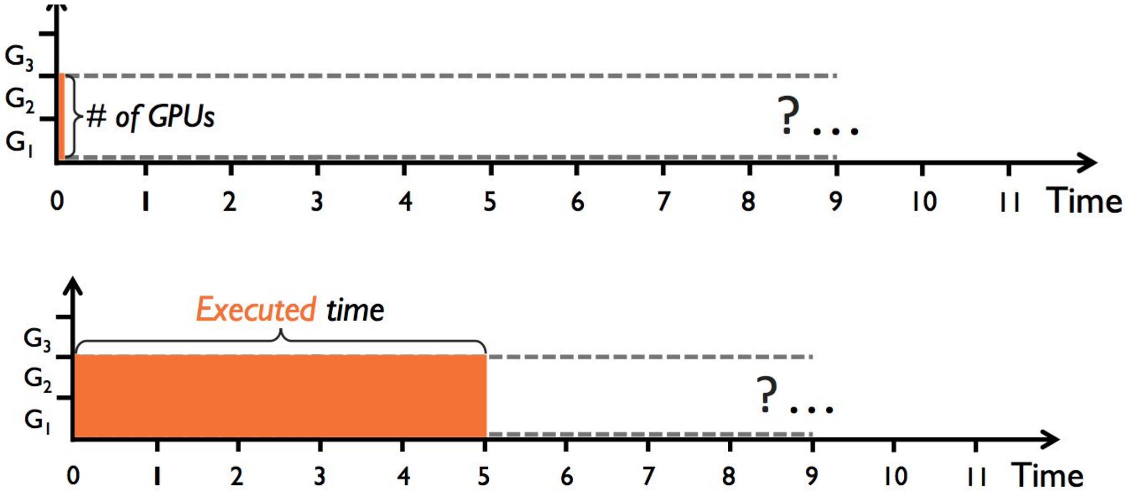
Tiresias - Approaches

1. Job Completion Time (Latency): Age based scheduler
   - Without knowing complete knowledge of jobs

2. Job Placement (Resource Utilization): Model Profile-Based Placement
   - Place jobs without additional info from users

Characteristics of DL Training Jobs

- Consider # of GPUs (spatial) and executed time (temporal)
- Attained Service (Age) = # GPUs * Executed Time



Solution I:Two-Dimensional Age-Based Scheduler

- Least-Attained Service (LAS)
  - Prioritize job that has the **shortest executed time**
- Gittins Index policy
  - Need the distribution of job execution time
  - Prioritize job that has the highest probability to complete in the near future
- Age calculated by two-dimensional attained service
  - i.e., a job's total executed GPU time (# of GPUs × executed time)
- No Prior Info about Jobs
  - Use LAS
- With Partial Info (distribution of job GPU time)
  - 2D-Gittins Index

2D-Gittins Index: Example

- Higher ***probability to complete*** (***Gittins Index***), higher priority

	# of GPUs	Execution time
J_1	2	2
J_2	1	8
J_3	2	6

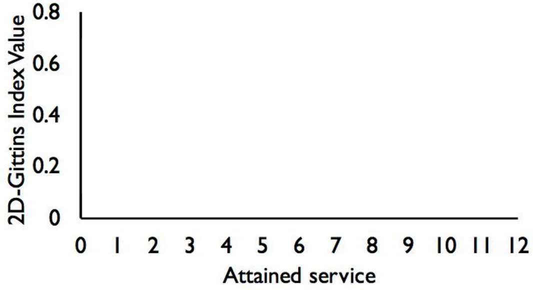
Not Known by the Scheduler

*Slide credit: Juncheng Gu, original author of the paper

2D-Gittins Index: Example

- Higher ***probability to complete*** (***Gittins Index***), higher priority

	# of GPUs	Distribution	Attained Service
J_1	2	2	0
J_2	1	(4, 8, 12)	0
J_3	2	6	0



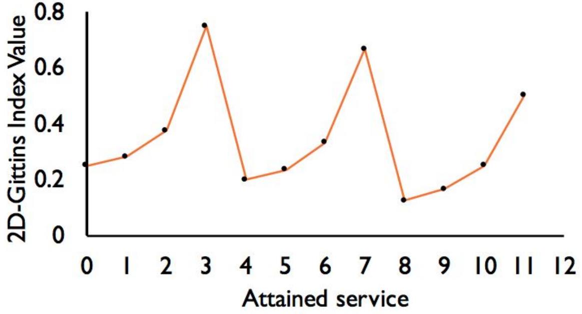
The distribution of job total GPU time may be obtained from cluster history log

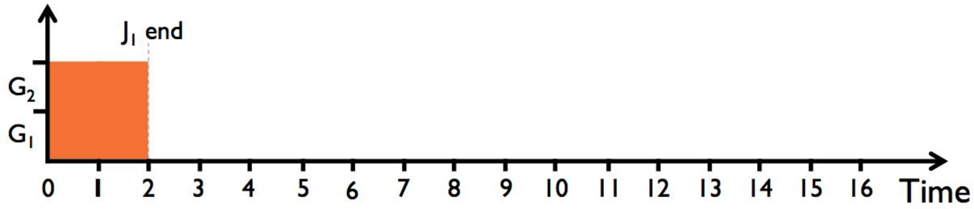
*Slide credit: Juncheng Gu, original author of the paper

2D-Gittins Index: Example

- Higher ***probability to complete*** (***Gittins Index***), higher priority

	# of GPUs	Distribution	Attained Service	Gittins Index
J_1	2	2	4	0.2
J_2	1	(4, 8, 12)	0	0.25
J_3	2	6	0	0.25



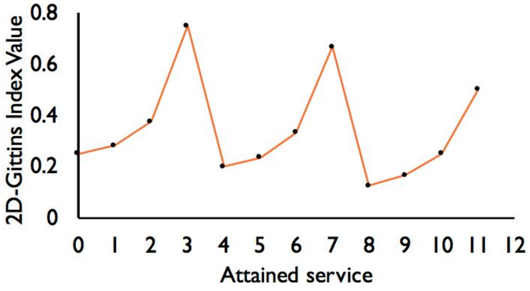


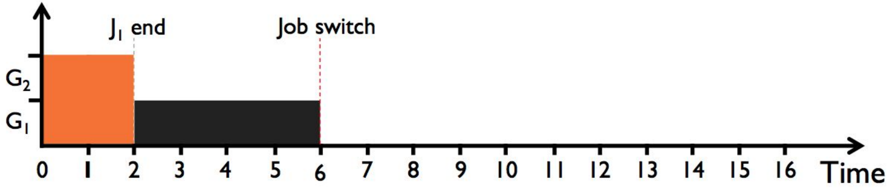
*Slide credit: Juncheng Gu, original author of the paper

2D-Gittins Index: Example

- Higher ***probability to complete*** (***Gittins Index***), higher priority

	# of GPUs	Distribution	Attained Service	Gittins Index
J_1	2	2	4	0.2
J_2	1	(4, 8,12)	4	0.2
J_3	2	6	0	0.25

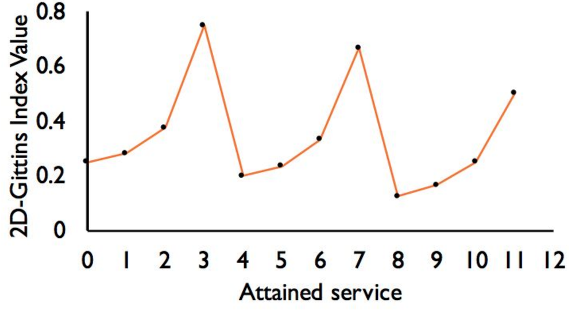


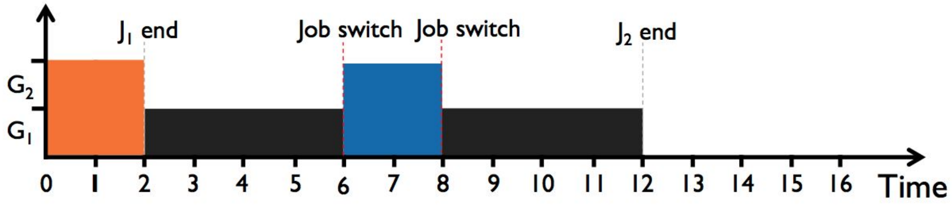


*Slide credit: Juncheng Gu, original author of the paper

2D-Gittins Index: Example

	# of GPUs	Distribution	Attained Service	Gittins Index
J_1	2	2	4	0.2
J_2	1	(4, 8, 12)	8	0.125
J_3	2	6	4	0.2

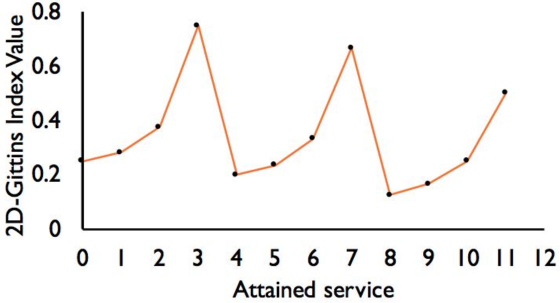


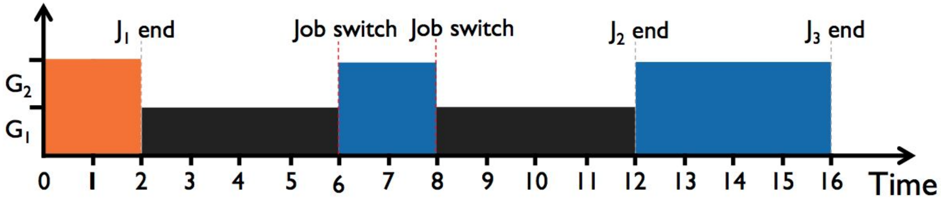


*Slide credit: Juncheng Gu, original author of the paper

2D-Gittins Index: Example

	# of GPUs	Distribution	Attained Service	Gittins Index
J_1	2	2	4	0.2
J_2	1	(4, 8, 12)	8	0.125
J_3	2	6	12	N/A



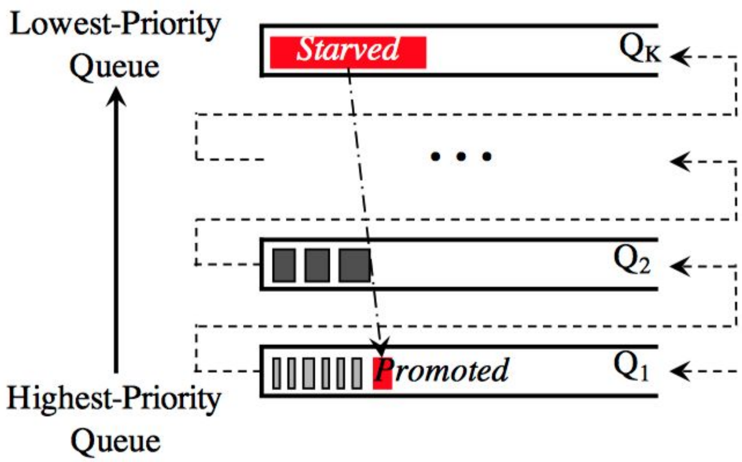


	Extra Information	Avg. JCT
2D-Gittins Index	GPU time distribution	10.0
2D-LAS	None	11.7

*Slide credit: Juncheng Gu, original author of the paper

Avoid Frequent Preemption/Starvation

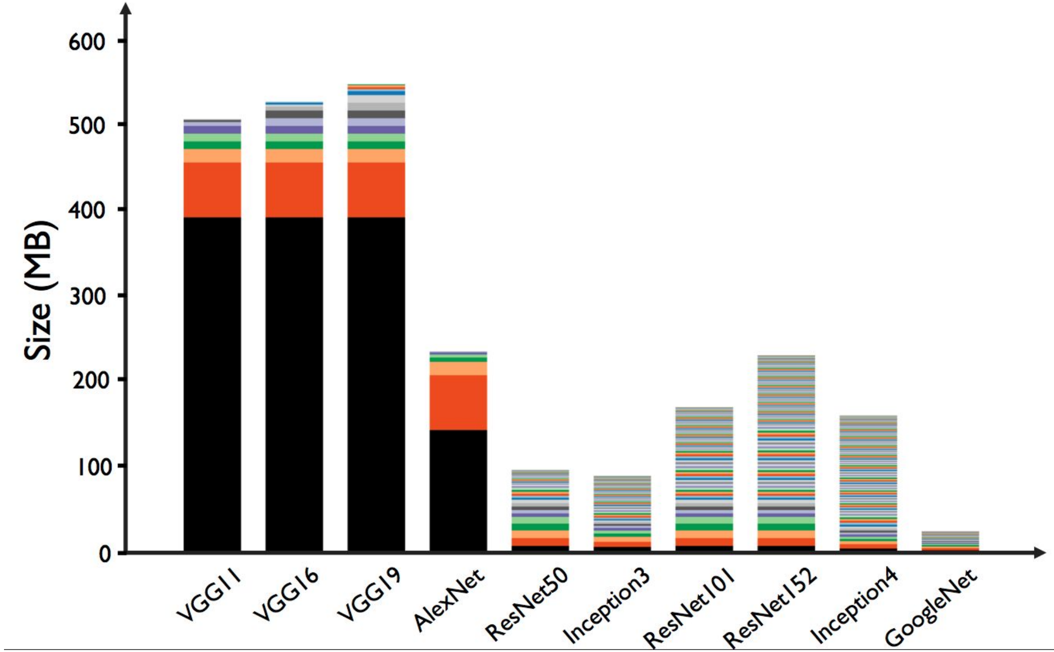
- Job switch are expensive
- Discretized continuous priority to logical queues
- To avoid starvation, promote the job waiting time > threshold



# of Queues: K=2 for the paper

Solution II: Profiling Model Characteristics

- Large tensors in models cause network contention in distributed training

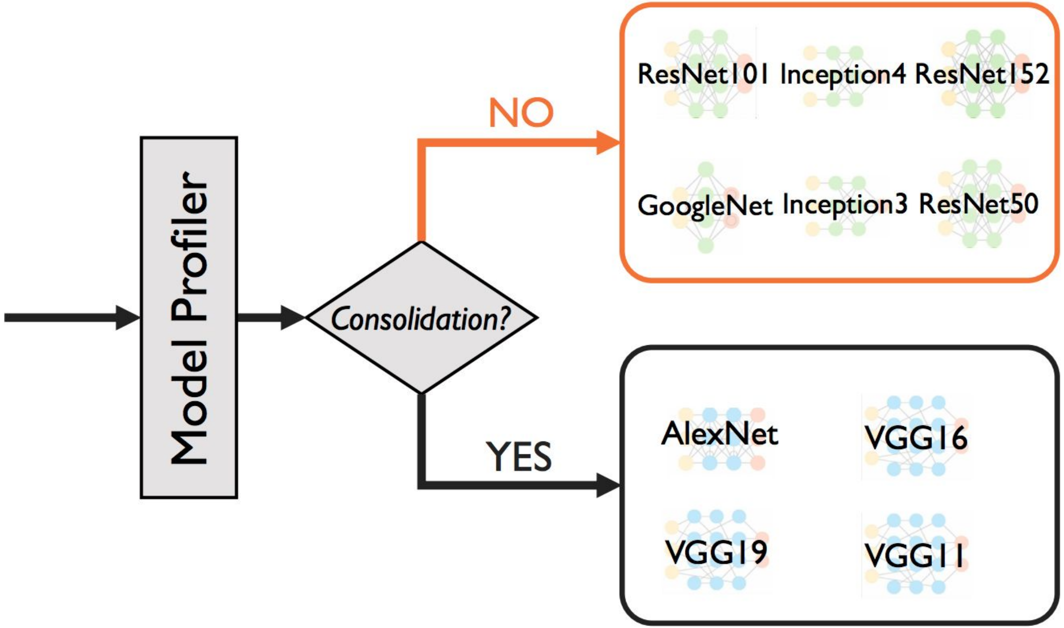


Consolidated placement is needed when the model is **highly skewed** in its tensor size

Model Profiler

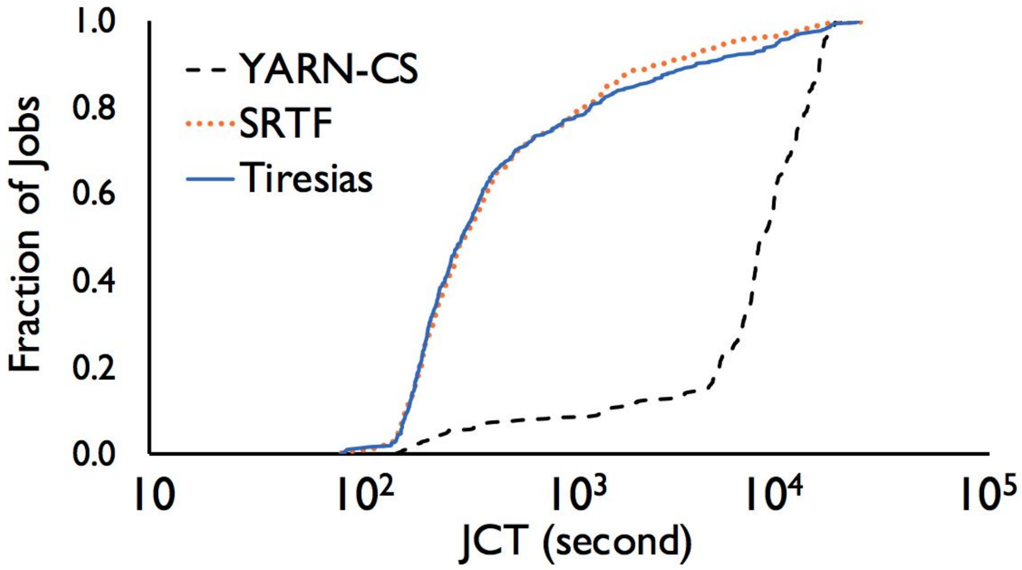
- Identify the amount of skew in tensor distributions

- Determine whether the job needs consolidation based on the profiler result



Evaluation - Testbed & Traces

- 60 GPU cluster & Traces from Microsoft



Baseline: YARN-CS used by Microsoft

5.5x Avg. JCT improvement (w.r.t. YARN-CS)

Comparable performance to SRTF

Evaluation - Sources of Improvements

- Where are the improvements from?

Performance gain from job placement

	Average	Median	95th
YARN-CS	8146s	7464s	15327s
SRTF	593s	32s	3133s
Tiresias-G	1005s	39s	7933s
Tiresias-L	963s	13s	7755s

Reducing queuing delay

Limitations & Future Work

- Lack of formal analysis
  - Configurations of best parameters vary across clusters
- Lightweight preemption methods can help
- Fine-grained job placement
  - Current method avoid network transfers
  - But there can be interference within a server (like PCI bus)

Discussion

Question from piazza:

I wonder, is it possible to use Tiresias with model parallelism or even pipeline parallelism? What was the main motivation for data parallelism?

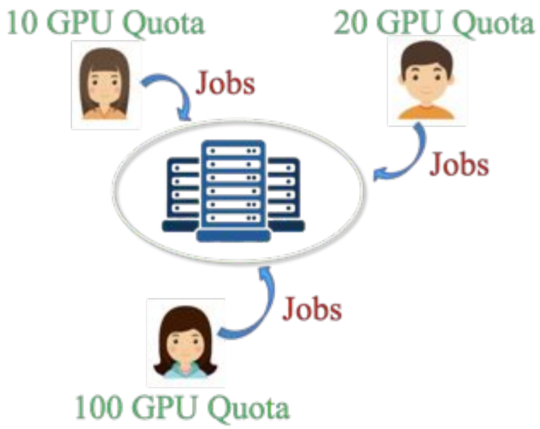
HiveD: Sharing a GPU Cluster for Deep Learning with Guarantees

Hanyu Zhao, et al.

Presenter: Shuowei Jin, Wenyuan Ma

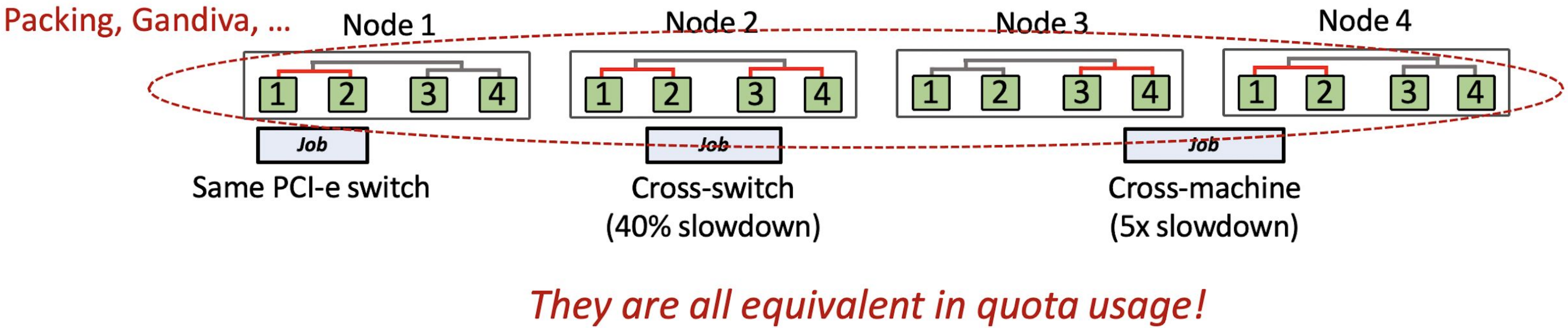
Introduction: Today's GPU Cluster

- Shared GPU Cluster
  - Multiple tenants
  - Current Resource Reservation Mechanism
    - Based on Quota(i.e. Number of GPUs)
    - Sharing Anomaly Problem



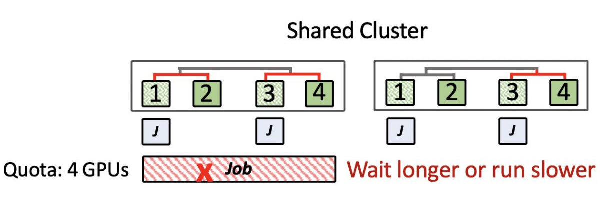
Background: Quota-based

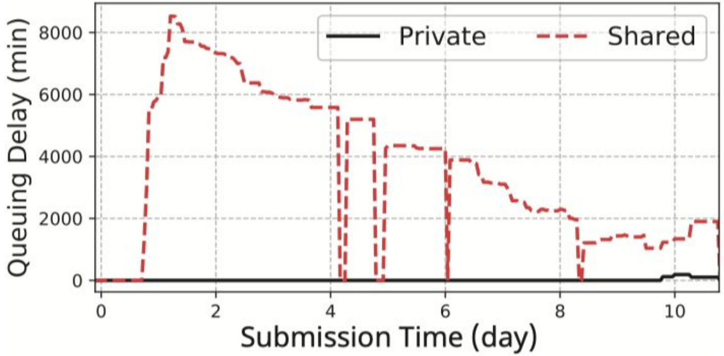
- GPU<->Tokens as quota -> Each Tenant's request
- Problem: Neglect the **affinity** factor
  - Low training speed



Background: Sharing Anomaly

- Sharing Anomaly Definition:
  - Private Cluster ✔️
  - Shared Cluster affinity requirement ❌
- External Fragmentation leads to the Sharing Anomaly
  - Users usually specifies the GPU Requirements(8 nodes each with 8 GPUs)
    - Hard Requirement: wait in queue
    - Soft Requirement: scheduled 64 nodes each with 1 GPUs, based on quota
- Solving Approach:
  - Propose HiveD
  - Resource Reservation Framework
  - Focus on eliminating sharing anomaly





HiveD

Focus on how to **dynamically reserving** GPU considering the **affinity**

HiveD

Focus on how to **dynamically reserving** GPU considering the **affinity**

Sounds familiar in **Memory**?

Inspirations from OS (Personal Guess)

HiveD

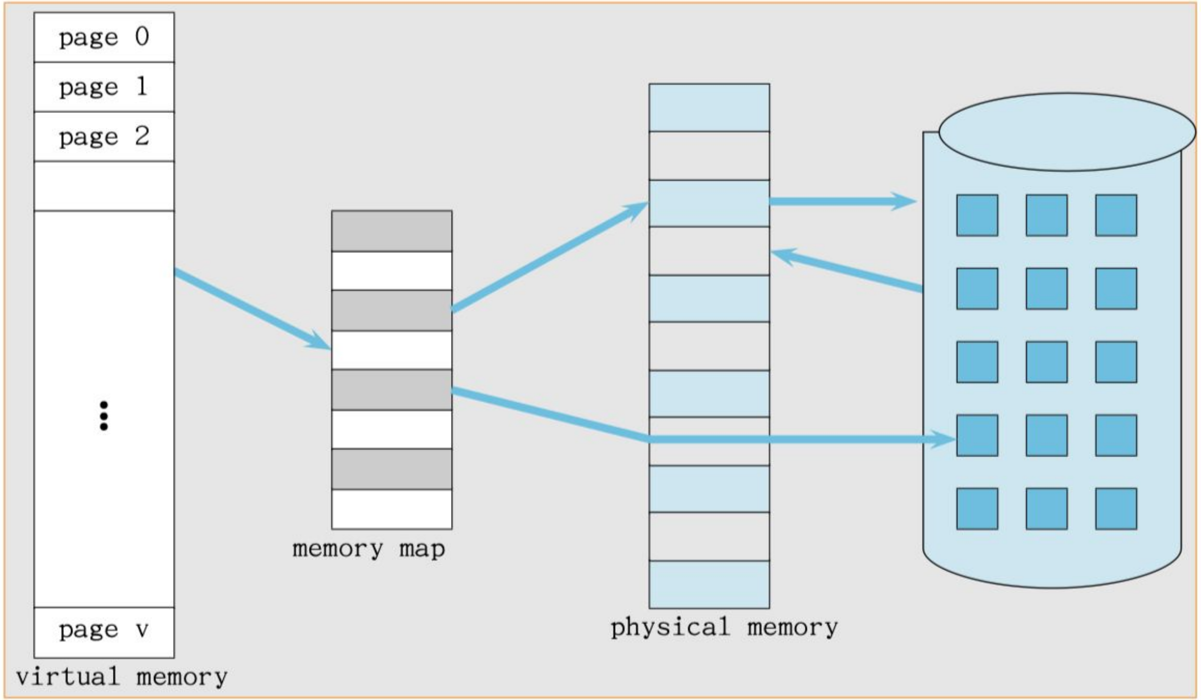
- GPU affinity
  - 8-GPU job runs way faster on one node than on eight nodes
  - Close GPU helps
- GPU's sharing Anomaly (External Fragmentation)
  - Virtual Private Cluster
  - Buddy Cell Allocation Algorithm

Operating System

- Spatial Locality
  - Once a location is referenced, its nearby locations will be referenced soon
  - Continuous Memory helps
- Memory External Fragmentation Problem
  - Virtual Memory
  - Buddy Algorithms in Linux to solve the memory fragmentation problem

Recap: Virtual Memory in OS

A continuous virtual address space can help programmers a lot



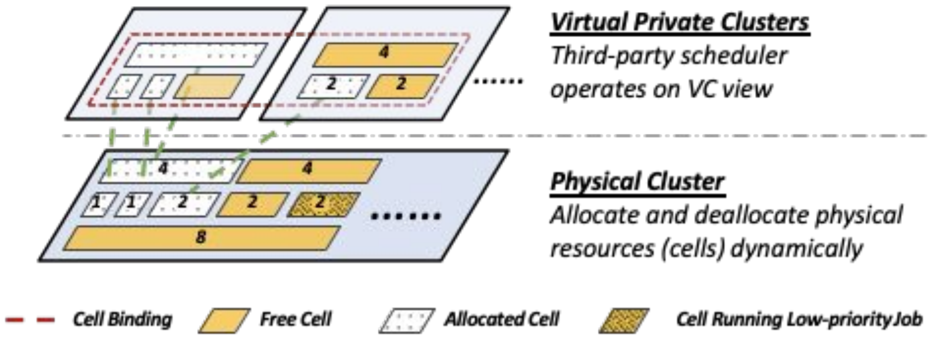
HiveD's System Overview

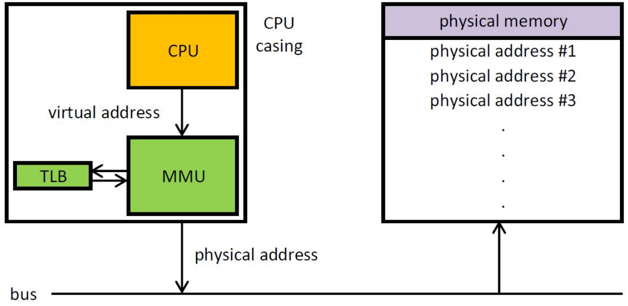
HiveD

- Virtual Private Cluster
  - Logical Cell Abstraction
  - Compatibility to third-party
- Virtual to Physical
  - Dynamic Allocation

OS

- Virtual Memory
  - Logical Memory Abstraction
  - Compatibility to applications
- Virtual to Physical
  - Dynamic Memory Allocation





CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

HiveD's Cell Structures

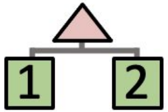
A cell is a set of GPUs at a certain level of affinity

Level-1 cell: GPU 1

*Slide credit: Hanyu Zhao, original author of the paper

HiveD’s Cell Structures

A cell is a set of GPUs at a certain level of affinity

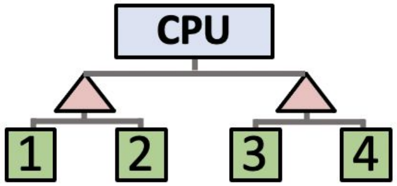
Level-2 cell: PCI-e switch 

*Slide credit: Hanyu Zhao, original author of the paper

HiveD's Cell Structures

A cell is a set of GPUs at a certain level of affinity

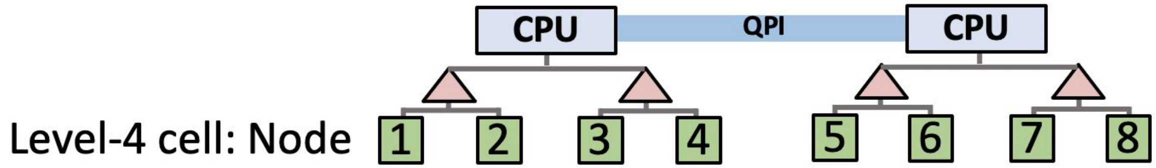
Level-3 cell: CPU socket



*Slide credit: Hanyu Zhao, original author of the paper

HiveD's Cell Structures

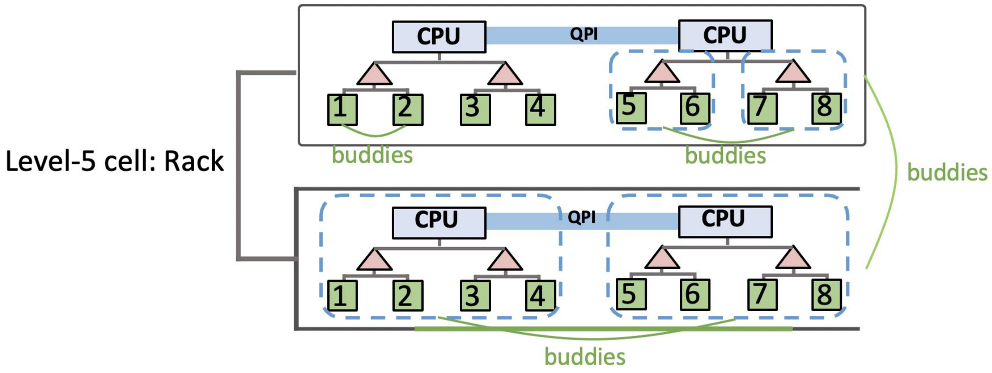
A cell is a set of GPUs at a certain level of affinity



*Slide credit: Hanyu Zhao, original author of the paper

HiveD's Cell Structures

A cell is a set of GPUs at a certain level of affinity

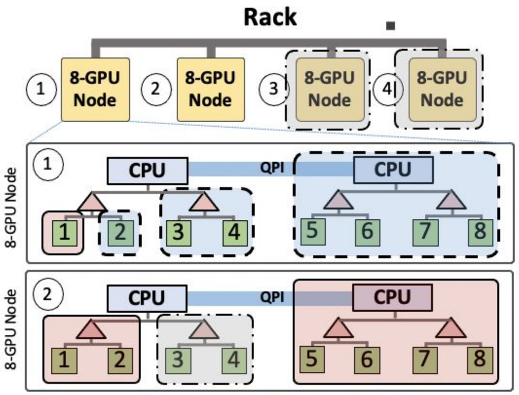


A cell can be split into multiple equivalent **buddy cells**

*Slide credit: Hanyu Zhao, original author of the paper

Comparison

HiveD's Cell Structures



Dynamic binding via Buddy Cell Allocation

Tenant A

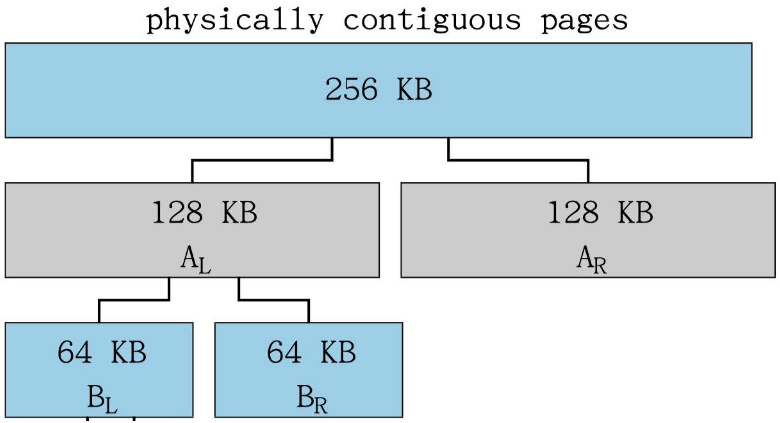
Tenant B

Tenant C

Cell Level	A	B	C
L4 cell (8-GPU)	0	0	2
L3 cell (4-GPU)	1	1	0
L2 cell (2-GPU)	1	1	1
L1 cell (1-GPU)	1	1	0

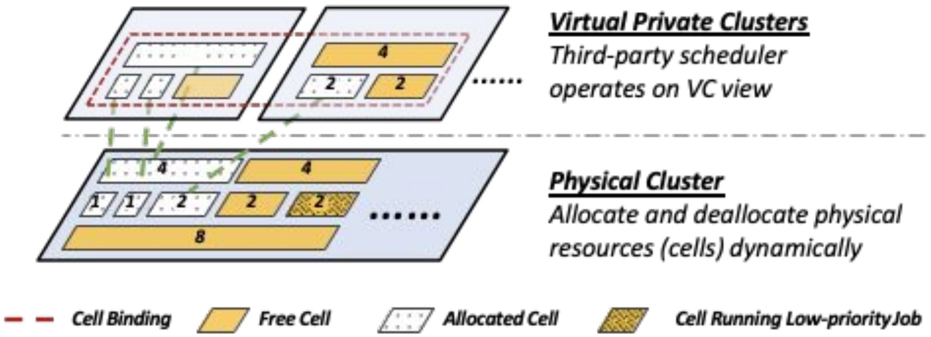
VC Assignment

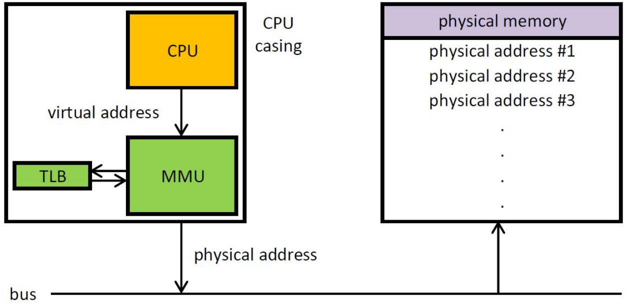
OS's Physical Memory Structures



HiveD's System Overview

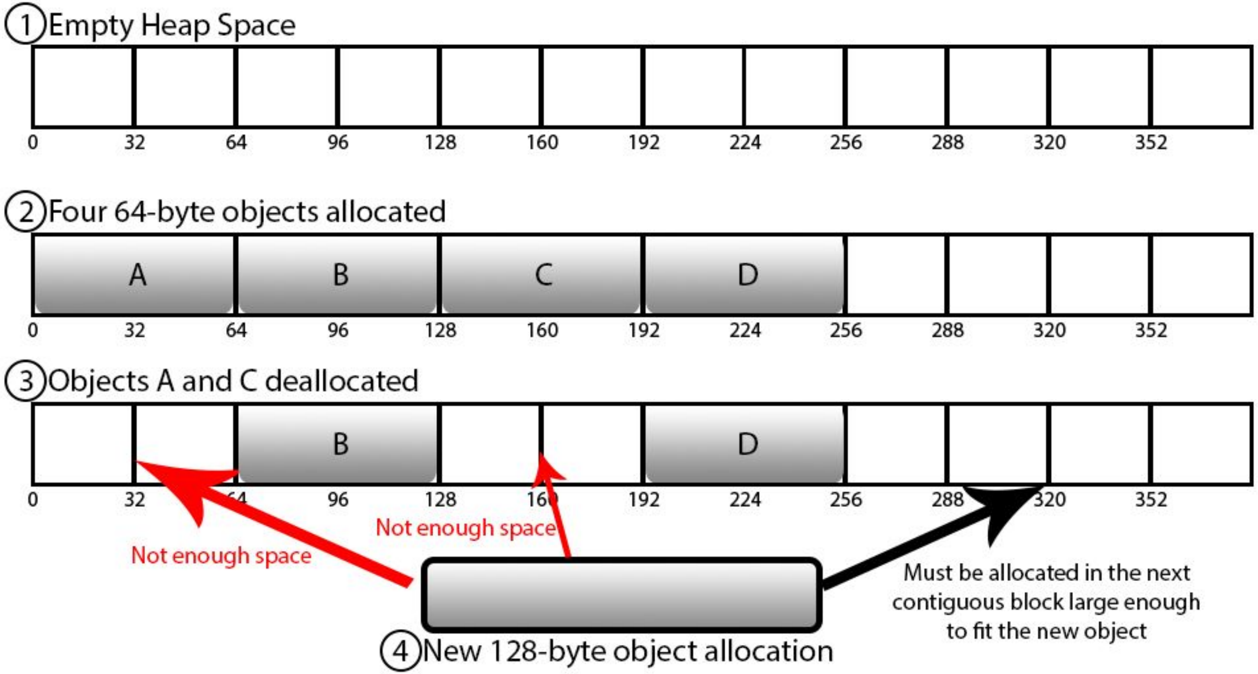
HiveD		OS
• Virtual Private Cluster – Logical Cell – Compatibility to third-party	⟷	• Virtual Memory – Logical Memory – Compatibility to applications
• Virtual to Physical – Dynamic Allocation	⟷	• Virtual to Physical – Dynamic Memory Allocation





CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

Recap: Memory Fragmentation



*Picture credit: Unity 2017 Game Optimization - Second Edition by Chris Dickinson

Recap: Buddy Algorithm in Linux

- If memory is to be allocated
    1. Look for a memory slot of a suitable size (the minimal 2^k block that is larger or equal to that of the requested memory)
        1. If it is found, it is allocated to the program
        2. If not, it tries to make a suitable memory slot. The system does so by trying the following:
            1. Split a free memory slot larger than the requested memory size into half
            2. If the lower limit is reached, then allocate that amount of memory
            3. Go back to step 1 (look for a memory slot of a suitable size)
            4. Repeat this process until a suitable memory slot is found
- If memory is to be freed
    1. Free the block of memory
    2. Look at the neighboring block – is it free too?
    3. If it is, combine the two, and go back to step 2 and repeat this process until either the upper limit is reached (all memory is freed), or until a non-free neighbour block is encountered

Recap: Buddy Algorithm in Linux

Step1. Initial Situation

<table>
<tr><td>Step</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td colspan="16">16KB</td></tr>
</table>

Recap: Buddy Algorithm in Linux

Step2. Request Memory of 1KB

Step																
1	16KB															
2.1	8KB								8KB							

Recap: Buddy Algorithm in Linux

Step2. Request Memory of 1KB

<table>
<tr><td>Step</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td colspan="16">16KB</td></tr>
<tr><td>2.1</td><td colspan="8">8KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.2</td><td colspan="4">4KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
</table>

Recap: Buddy Algorithm in Linux

Step2. Request Memory of 1KB

<table>
<tr><td>Step</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td colspan="16">16KB</td></tr>
<tr><td>2.1</td><td colspan="8">8KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.2</td><td colspan="4">4KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.3</td><td colspan="2">2KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
</table>

Recap: Buddy Algorithm in Linux

Step2. Request Memory of 1KB

<table>
<tr><td>Step</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td colspan="16">16KB</td></tr>
<tr><td>2.1</td><td colspan="8">8KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.2</td><td colspan="4">4KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.3</td><td colspan="2">2KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
</table>

Recap: Buddy Algorithm in Linux

Step2. Request Memory of 1KB

<table>
<tr><th>Step</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></tr>
<tr><td>1</td><td colspan="16">16KB</td></tr>
<tr><td>2.1</td><td colspan="8">8KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.2</td><td colspan="4">4KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.3</td><td colspan="2">2KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.4</td><td>1KB</td><td>1KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
</table>

Recap: Buddy Algorithm in Linux

Step3. Request Memory of 2KB

<table>
<tr><td>Step</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td colspan="16">16KB</td></tr>
<tr><td>2.1</td><td colspan="8">8KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.2</td><td colspan="4">4KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.3</td><td colspan="2">2KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.4</td><td>1KB</td><td>1KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>3</td><td>1KB</td><td>1KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
</table>

Recap: Buddy Algorithm in Linux

Step4. Deallocate Memory of 1KB

Step																
1	16KB															
2.1	8KB								8KB							
2.2	4KB				4KB				8KB							
2.3	2KB		2KB		4KB				8KB							
2.4	1KB	1KB	2KB		4KB				8KB							
3	1KB	1KB	2KB		4KB				8KB							
4	1KB	1KB	2KB		4KB				8KB							

Recap: Buddy Algorithm in Linux

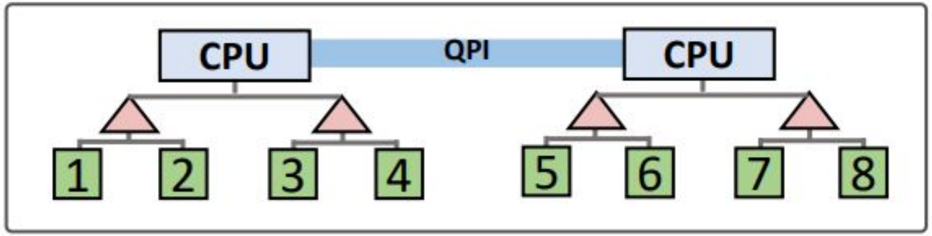
Step4. Deallocate Memory of 1KB

<table>
<tr><td>Step</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td colspan="16">16KB</td></tr>
<tr><td>2.1</td><td colspan="8">8KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.2</td><td colspan="4">4KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.3</td><td colspan="2">2KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>2.4</td><td>1KB</td><td>1KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>3</td><td>1KB</td><td>1KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
<tr><td>4</td><td colspan="2">2KB</td><td colspan="2">2KB</td><td colspan="4">4KB</td><td colspan="8">8KB</td></tr>
</table>

Buddy Cell Allocation Algorithm

To allocate a level-*k* cell

- If a free level-*k* cell is available, allocate one
- Otherwise, move up (*k*+1,*k*+2,...) until a higher level cell is available; split higher level cell recursively until free level-*k* cells are produced

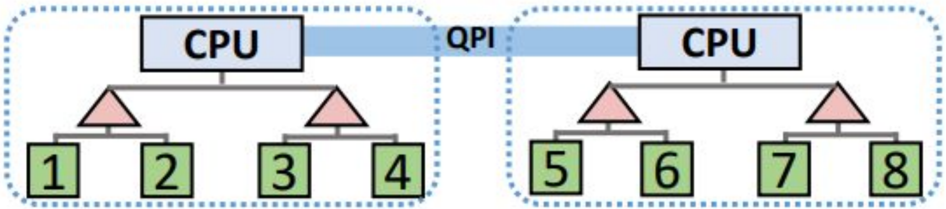


Cell request for level-1 (GPU)

Buddy Cell Allocation Algorithm

To allocate a level-*k* cell

- If a free level-*k* cell is available, allocate one
- Otherwise, move up (*k*+1,*k*+2,…) until a higher level cell is available; split higher level cell recursively until free level-*k* cells are produced

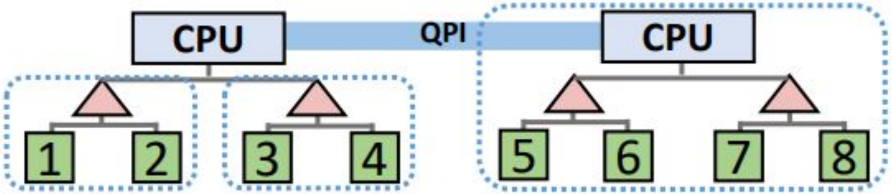


Cell request for level-1 (GPU)

Buddy Cell Allocation Algorithm

To allocate a level-*k* cell

- If a free level-*k* cell is available, allocate one
- Otherwise, move up (*k*+1,*k*+2,…) until a higher level cell is available; split higher level cell recursively until free level-*k* cells are produced

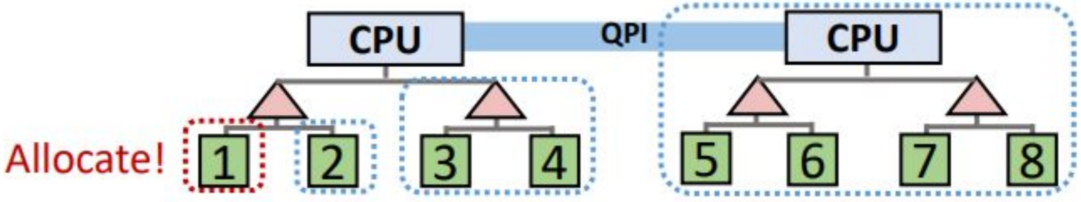


Cell request for level-1 (GPU)

Buddy Cell Allocation Algorithm

To allocate a level-*k* cell

- If a free level-*k* cell is available, allocate one
- Otherwise, move up (*k*+1,*k*+2,...) until a higher level cell is available; split higher level cell recursively until free level-*k* cells are produced



Cell request for level-1 (GPU)

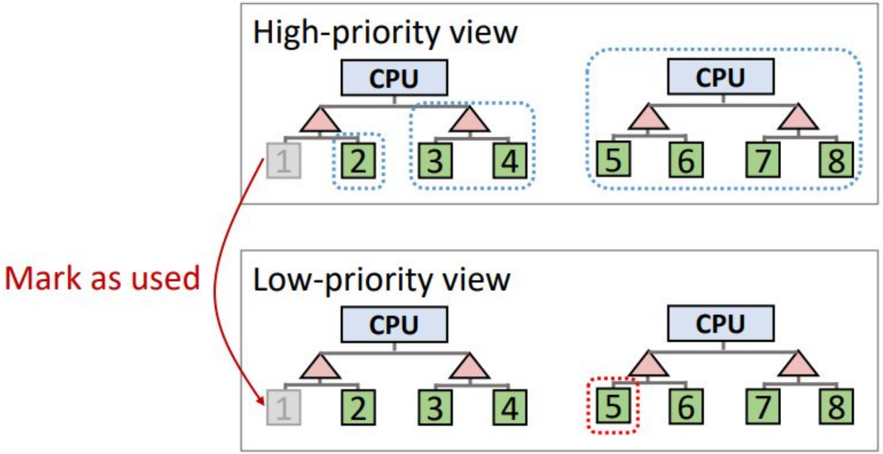
Buddy Cell Allocation Algorithm

- Releasing a level-k cell works oppositely
  - Add it to the free list of level-k
  - If all its buddy cells are free, merge them into a level-($k + 1$) cell
  - Continues recursively going up the levels
- Keep as many higher-level cells as possible
- Proven safety guarantee
  - Satisfies any cell request within a VC, if the initial VC assignment is feasible

Buddy Cell Allocation Algorithm

Support low-priority jobs to improve GPU utilization

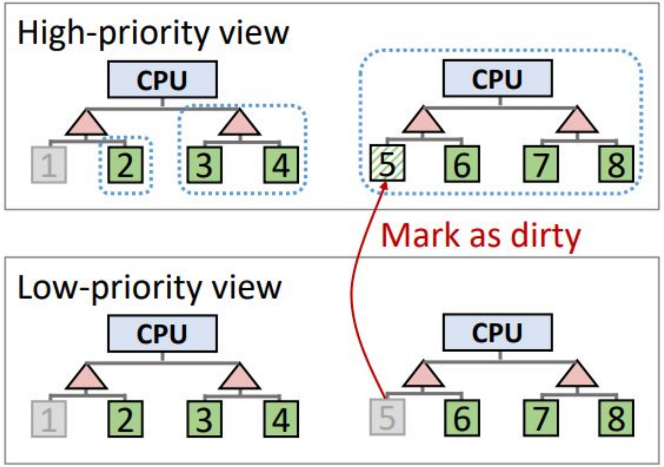
- Two cell views---one for high-priority (guaranteed) jobs, and one for low-priority opportunistic jobs
- Choose the farthest away cell to minimize preemption



Buddy Cell Allocation Algorithm

Support low-priority jobs to improve GPU utilization

- Two cell views---one for high-priority (guaranteed) jobs, and one for low-priority opportunistic jobs
- Choose the farthest away cell to minimize preemption



Implementation

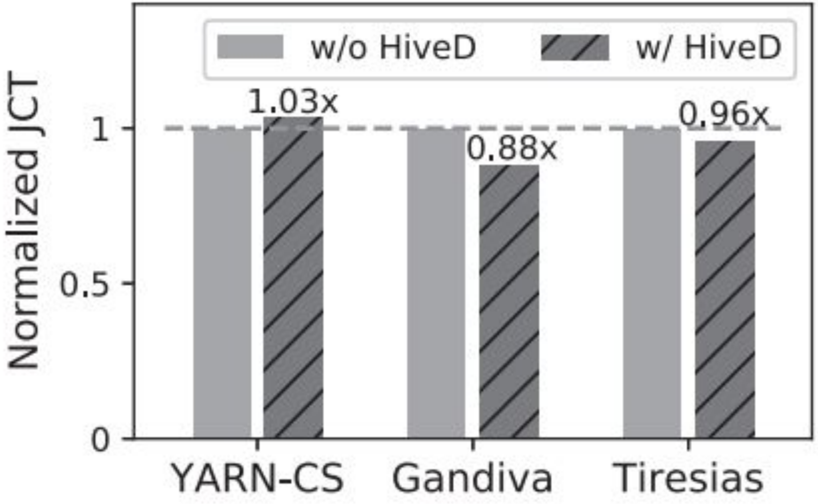
- Implemented on Kubernetes
- Integrated with Microsoft OpenPAI
- Fault Tolerance: dynamic binding avoids a faulty cell
- Experiment setup
  - 2-month trace from a production cluster with deep learning training jobs
  - 96-GPU cluster deployed on Azure, shared by 11 tenants

kubernetes

Open Platform for AI
(OpenPAI)

Evaluation: Completion Time

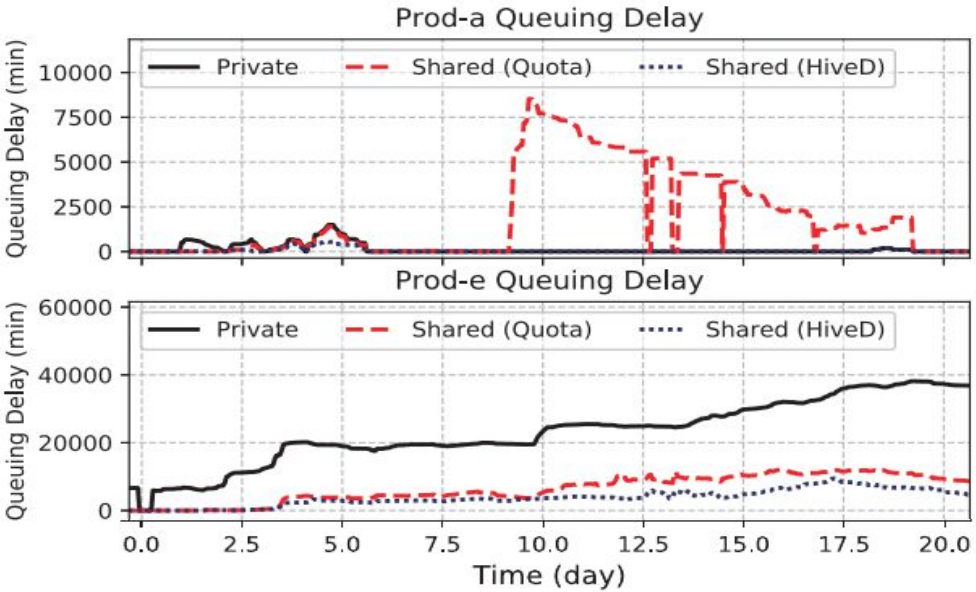
HiveD exhibits similar job completion time compared to those without HiveD



Average job completion time across all tenants

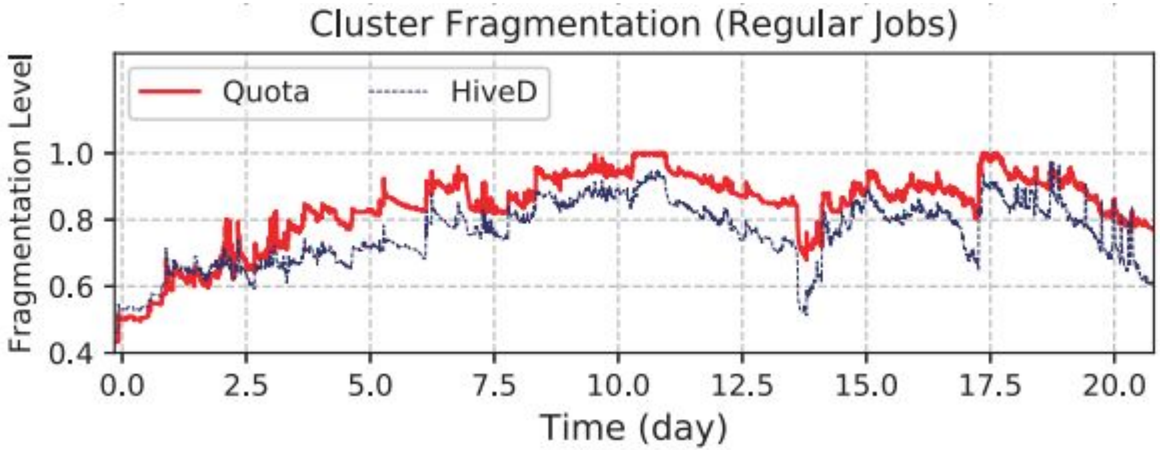
Evaluation: Queuing Delay

HiveD achieves the shortest queuing delay in two representative tenants, prod-a and prod-e



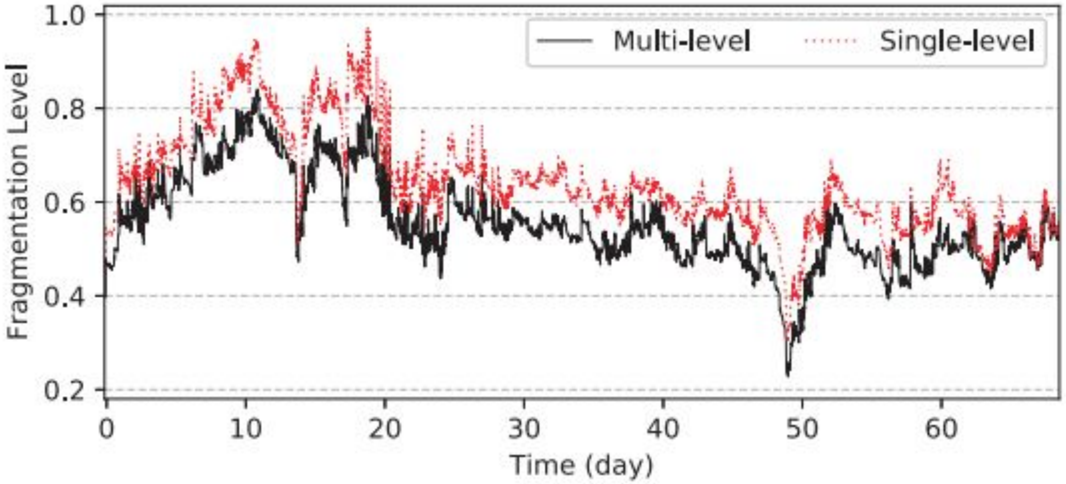
Evaluation: Level of Fragmentation

- Level of Fragmentation: proportion of 8-GPU nodes that cannot provide 8-GPU affinity for a high-priority job
- The fragmentation level in HiveD is lower than that in the quota-based scheme



Evaluation: Multi-Level Cells

The fragmentation level is always lower with multi-level cells



Fragmentation with multi- and single-level cells

Summary - Related works in the field

GPU cluster scheduling

- Optimus - Achieve dynamic resource allocation with est. of models [1]
- Gandiva - Utilize intra-job predictability to improve latency and efficiency of training [2]
- Themis - Provide fairness among jobs at each server [4]

Affinity-aware schedulers for deep learning training

- Tiresias, Gandiva[2], topology-aware placement algorithm [3]
- HiveD provides the VC abstraction for them and maps VC to physical clusters

[1] Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters, EuroSys 18

[2] Gandiva: Introspective Cluster Scheduling for Deep Learning, OSDI 18

[3] Topology-aware GPU scheduling for learning workloads in cloud environments, SC 17

[4] Kshiteej Mahajan, Arjun Singhvi, Arjun Balasubramanian, Varun Batra, Surya Teja Chavali, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient gpu cluster scheduling for machine learning workloads. USENIX NSDI, 2020.

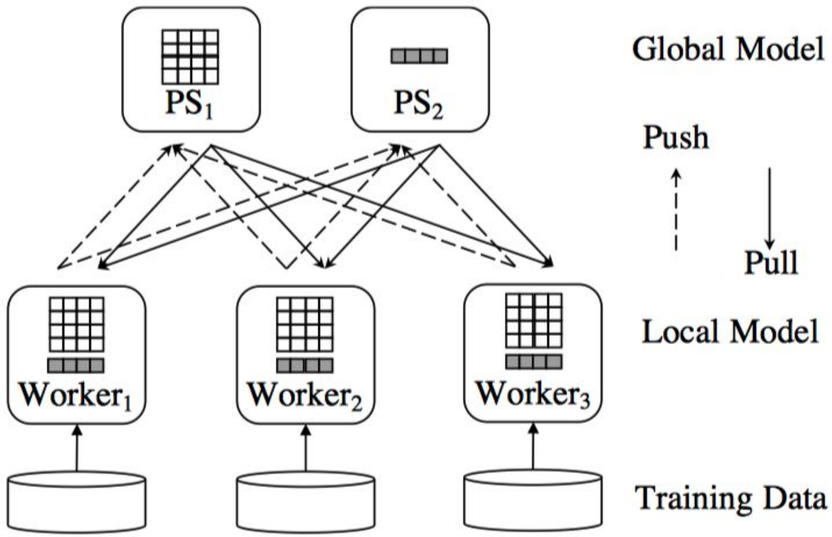
Discussion

Questions from Piazza:

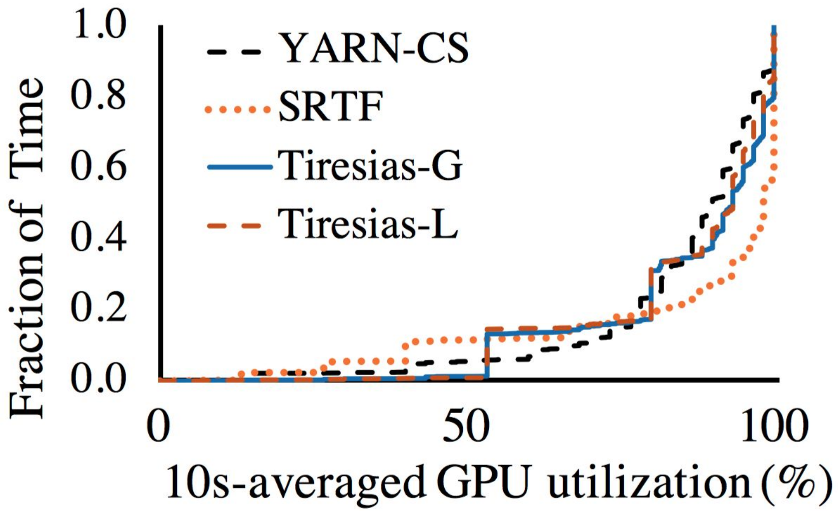
HiveD:

I'm don't quite understand why the Buddy Cell algorithm decreases cluster fragmentation. I feel like the regular reservation system is better with fragmentation since it doesn't care about affinity and will just give the tenant the number of GPUs it asked for. In fact, I think that affinity and fragmentation are almost trade offs, but somehow the buddy cell algorithm gets both.

Backup slides



Evaluation - Resource Utilization (Backup)



Resource Utilization

Evaluation - Trace Driven Simulation (Backup)

- Job traces from Microsoft

2x performance improvement over Gandiva (OSDI 18)