14th USENIX Symposium on Operating Systems Design and Implementation (**OSDI '20**)

HiveD:
Sharing a GPU Cluster for Deep Learning with Guarantees

Hanyu Zhao [1,3*], Zhenhua Han [2,3*], Zhi Yang [1], Quanlu Zhang [3], Fan Yang [3], Lidong Zhou [3], Mao Yang [3], Francis C.M. Lau [2], Yuqi Wang [3], Yifan Xiong [3], Bin Wang [3]

[1] *Peking University,* [2] *The University of Hong Kong,* [3] *Microsoft*

* Equal contributions

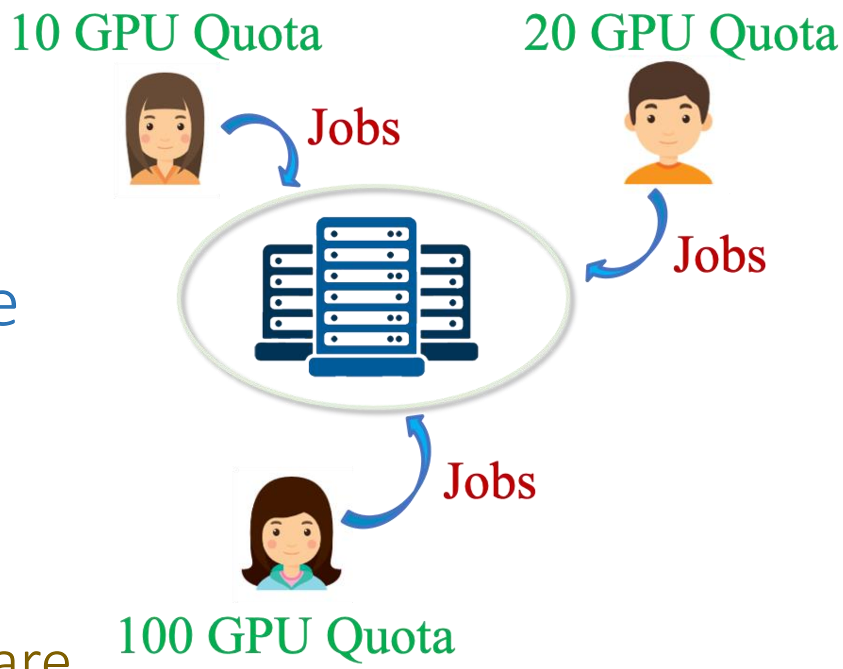
北京大學 PEKING UNIVERSITY

香 港 大 學 THE UNIVERSITY OF HONG KONG

Microsoft

Multi-Tenant GPU Clusters Today

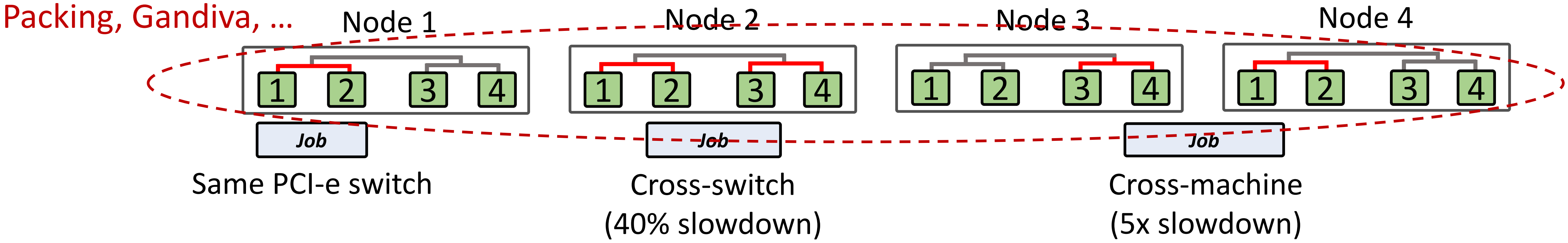
- Shared by multiple tenants
  - Built with budgets/hardware from tenants
- **Reservation** is necessary for *guaranteed* resource availability and user experiences
- Reserve ***number of GPUs (quota)*** for each tenant
  - A tenant expects to access at least its contributed share



Affinity Matters, but NOT Reserved

- Jobs usually have **affinity** requirements

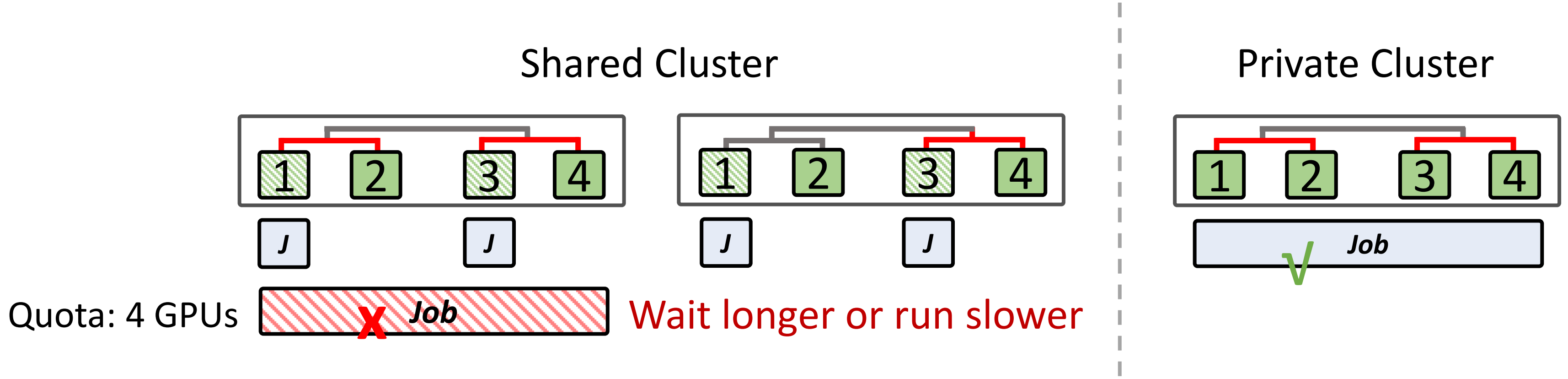
- Quota reservation + global affinity optimization (defragmentation)



They are all equivalent in quota usage!

Anomaly: Sharing Leads to Suffering!

- "*External*" fragmentation across tenants
  - Fragmentation from *other* tenants, even everyone is within the quota
  - Quota cannot *isolate* fragmentation across tenants
- External fragmentation makes sharing harmful to tenants
  - Worse performance in the *shared* cluster than in *private* clusters
  - Global defrag might sometimes hurt job performance (a complex multi-objective optimization)
- Real users are reverting to private clusters!



HiveD

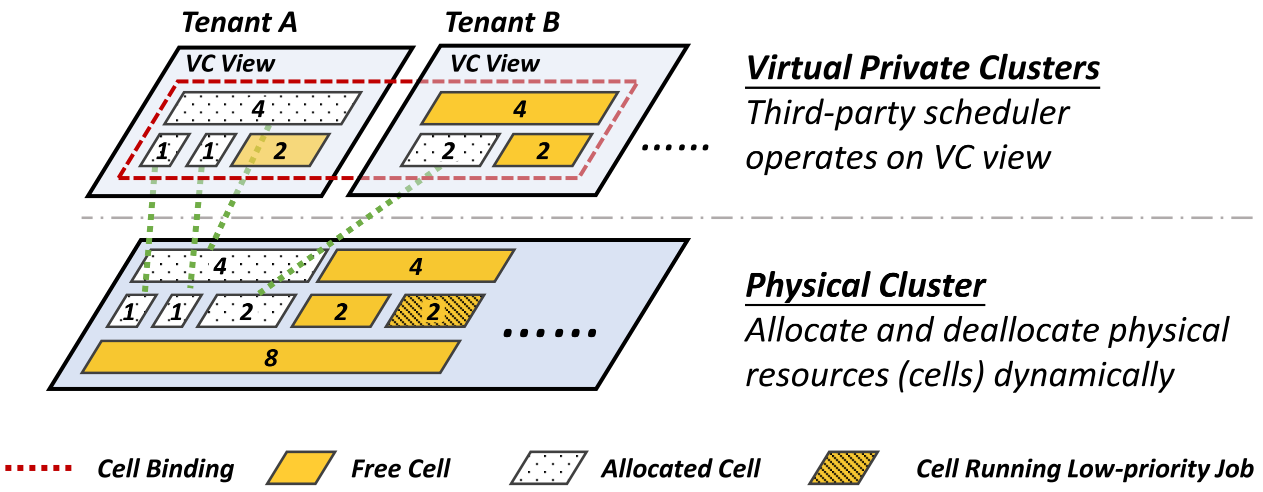
- Primary goal: **Sharing Safety**
  - Any sequence of GPU requests (possibly with affinity constraints) can be *satisfied on the shared cluster if satisfied on the tenant's private cluster*

- Key idea
  - ***Reserve GPU affinity explicitly***
  - Separate the concern of sharing safety from other scheduling goals

Two-Layer Architecture

- Virtual Private Cluster
  - New resource abstraction: *cell*, captures quota **and** affinity
  - Compatibility with SOTA deep learning schedulers in VCs
- From Virtual to Physical
  - Dynamic cell allocation with *proven sharing safety*
  - Natural support for low-priority jobs

Separation of Concerns	
Sharing Safety	✓
Scheduling Efficiency	✓
Utilization	✓



Two-Layer Architecture

- Virtual Private Cluster
  - New resource abstraction: *cell*, captures quota **and** affinity
  - Compatibility with SOTA deep learning schedulers in VCs
- From Virtual to Physical
  - Dynamic cell allocation with *proven sharing safety*
  - Natural support for low-priority jobs

Separation of Concerns	
Sharing Safety	✓
Scheduling Efficiency	✓
Utilization	✓

- *Best of both worlds*
  - A private cluster: guaranteed resource availability regardless of other tenants
  - A shared cluster: more resources when available

Hierarchical Cell Structures

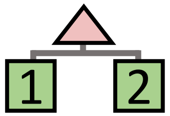
- A cell is a set of GPUs at a certain level of affinity

Level-1 cell: GPU $\boxed{1}$

Hierarchical Cell Structures

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

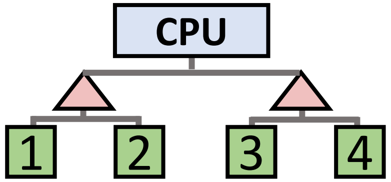
Level-2 cell: PCI-e switch



Hierarchical Cell Structures

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

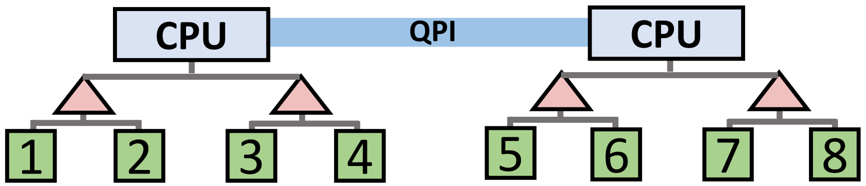
Level-3 cell: CPU socket



Hierarchical Cell Structures

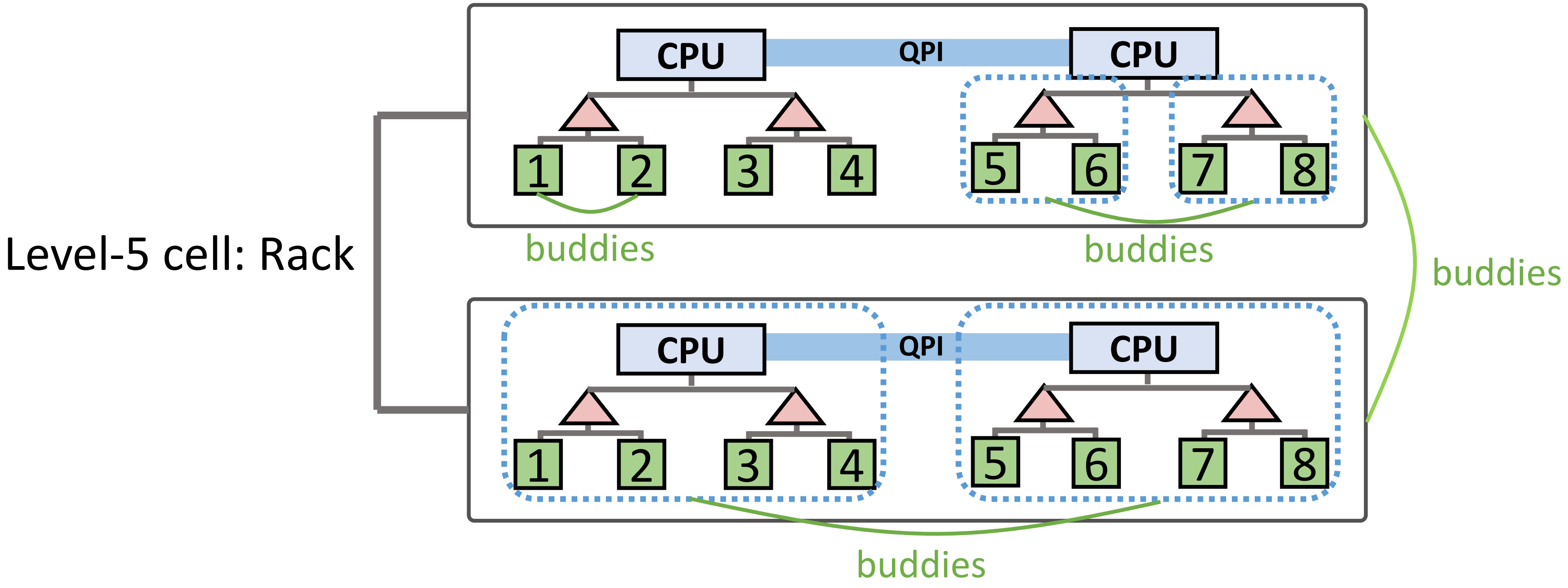
- A cell is a set of GPUs at a certain level of affinity

Level-4 cell: Node



Hierarchical Cell Structures

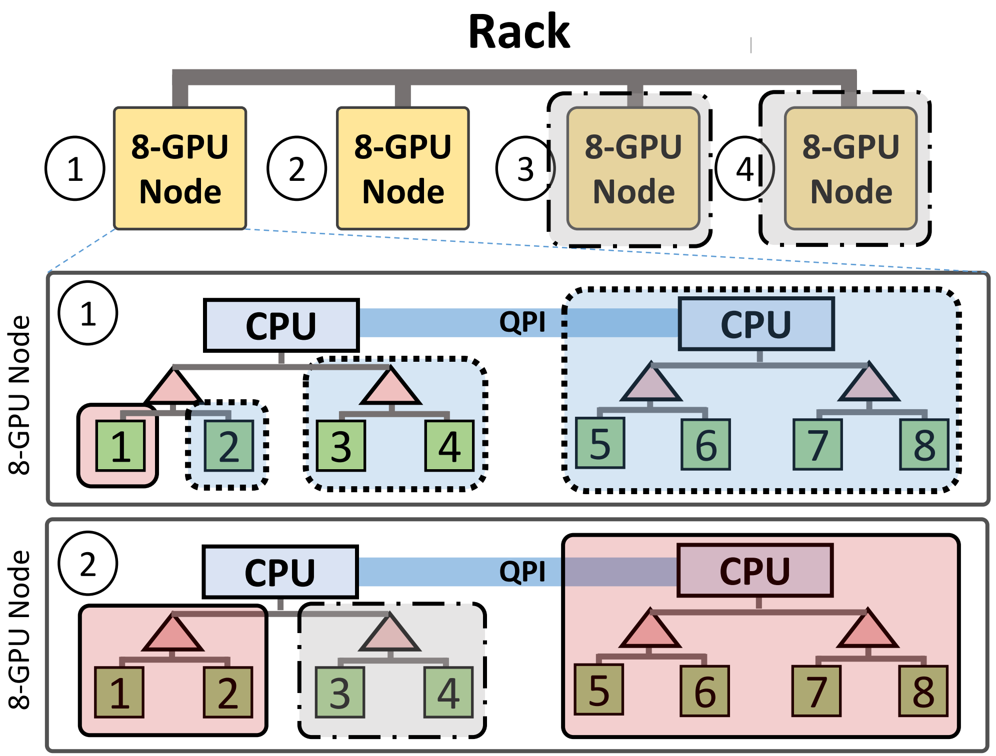
- A cell is a set of GPUs at a certain level of affinity



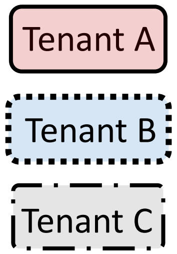
A cell can be split into multiple equivalent **buddy cells**

Virtual Private Clusters

- Cells at each level, modeling a tenant’s private cluster
- *Dynamic* cell binding
    - Reducing preemptions and fragmentation, handling faulty hardware
    - Handled by Buddy Cell Allocation algorithm



Dynamic binding via Buddy Cell Allocation

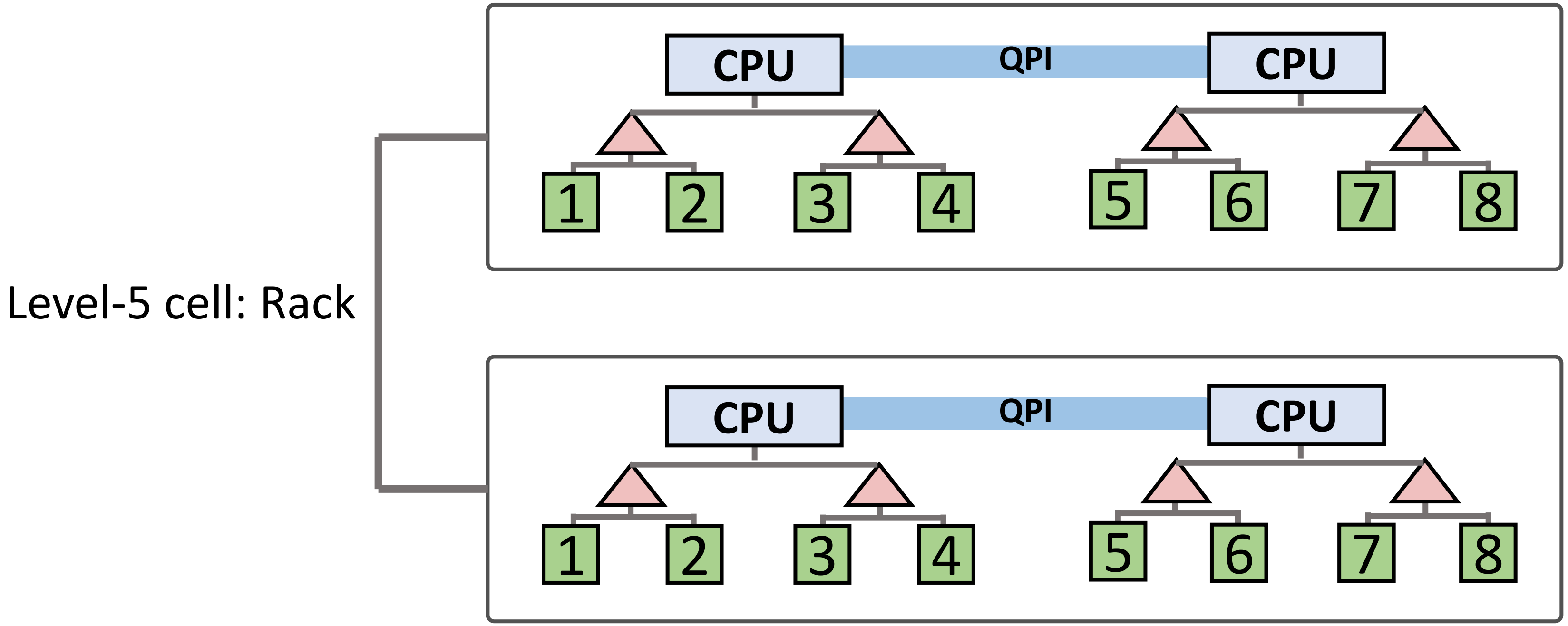


Cell Level	A	B	C
L4 cell (8-GPU)	0	0	2
L3 cell (4-GPU)	1	1	0
L2 cell (2-GPU)	1	1	1
L1 cell (1-GPU)	1	1	0

VC Assignment

Buddy Cell Allocation

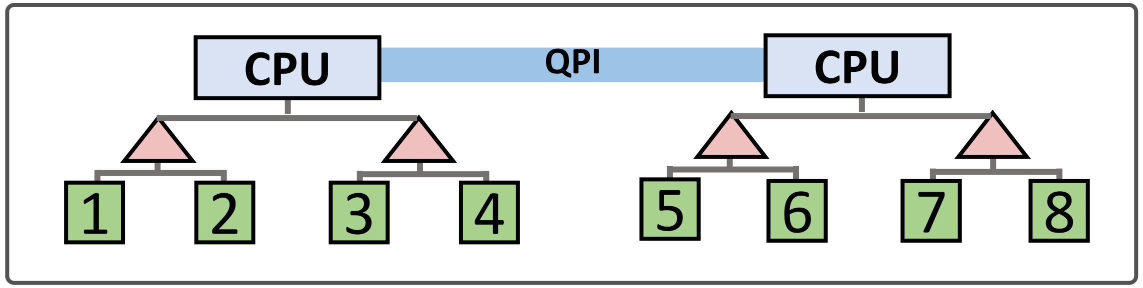
- For a cell request at level-k:
  - Allocate a free level-k cell if any
  - Split a free level-(k+1) cell otherwise



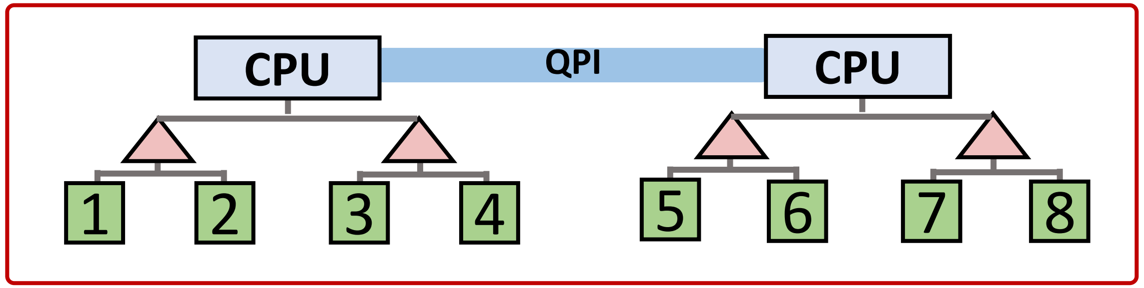
Cell request for level-4 (node)

Buddy Cell Allocation

- For a cell request at level-k:
  - Allocate a free level-k cell if any
  - Split a free level-(k+1) cell otherwise



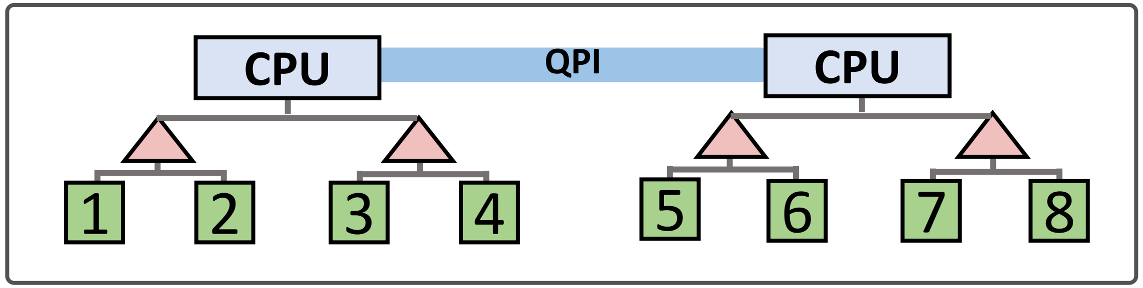
Allocate!



Cell request for level-4 (node)

Buddy Cell Allocation

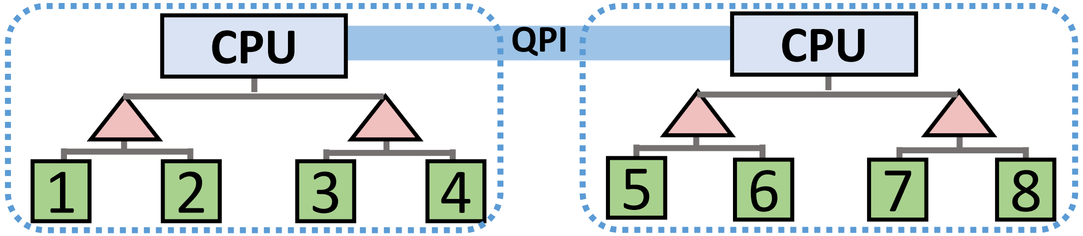
- For a cell request at level-k:
  - Allocate a free level-k cell if any
  - Split a free level-(k+1) cell otherwise



Cell request for level-1 (GPU)

Buddy Cell Allocation

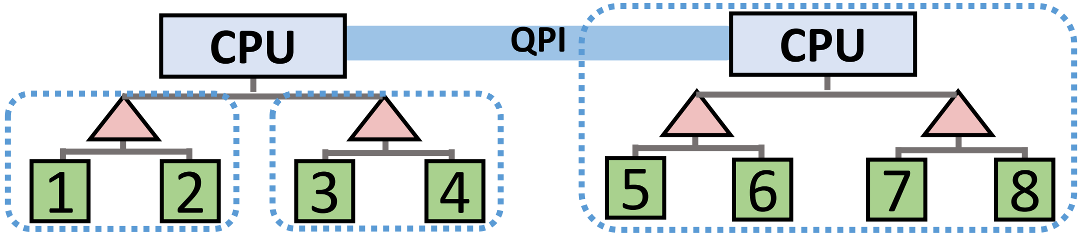
- For a cell request at level-k:
  - Allocate a free level-k cell if any
  - Split a free level-(k+1) cell otherwise



Cell request for level-1 (GPU)

Buddy Cell Allocation

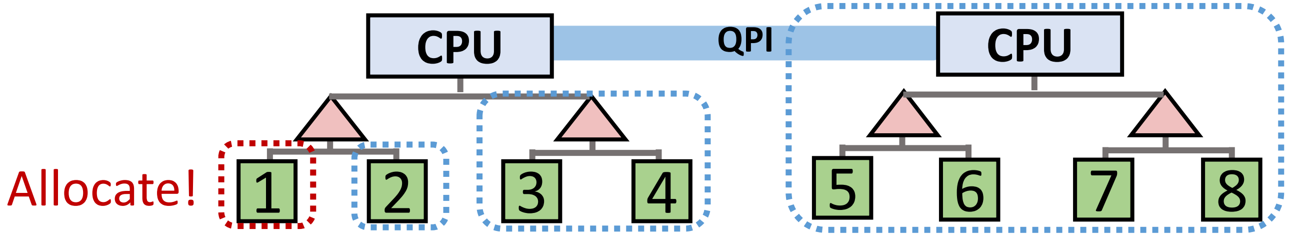
- For a cell request at level-k:
  - Allocate a free level-k cell if any
  - Split a free level-(k+1) cell otherwise



Cell request for level-1 (GPU)

Buddy Cell Allocation

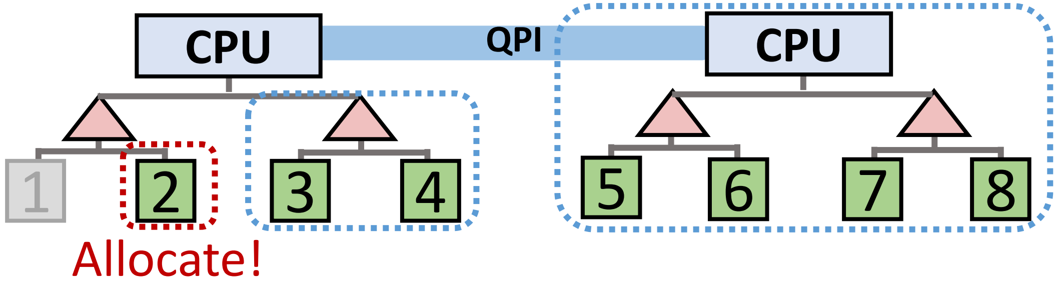
- For a cell request at level-k:
  - Allocate a free level-k cell if any
  - Split a free level-(k+1) cell otherwise



Cell request for level-1 (GPU)

Buddy Cell Allocation

- For a cell request at level-k:
  - Allocate a free level-k cell if any
  - Split a free level-(k+1) cell otherwise



Another Cell request for level-1 (GPU)

Buddy Cell Allocation

- For a cell request at level-k:
  - Allocate a free level-k cell if any
  - Split a free level-(k+1) cell otherwise

- Cell release (and merge) works oppositely

- 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

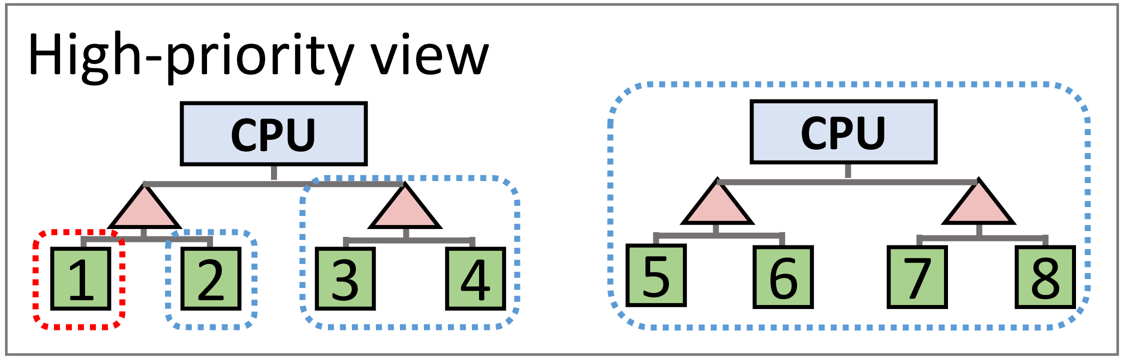
Algorithm 1 Buddy Cell Allocation Algorithm

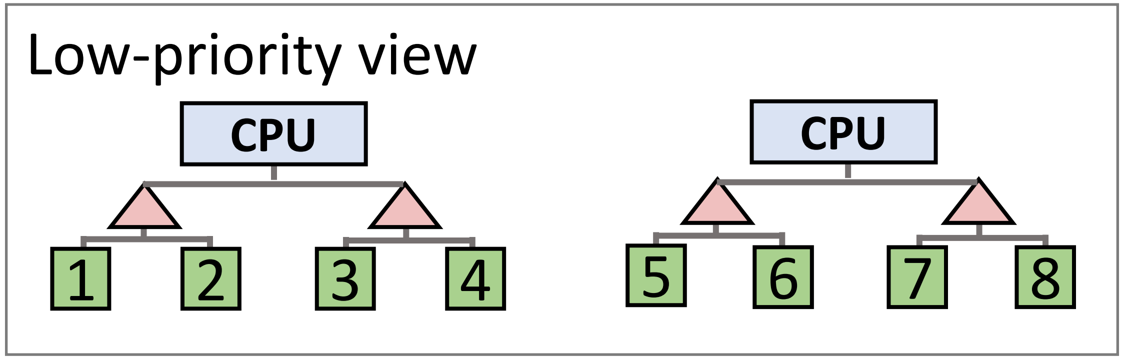
```
// Initial state of free_cells: only top level has cells
procedure ALLOCATECELL(cell_level)
  if free_cells[cell_level].size() == 0 then
    c = AllocateCell(cell_level+1)
    cells = Split(c)                         ▷ Split cells are buddies
    free_cells[cell_level].extend(cells)
  Return free_cells[cell_level].pop()

procedure RELEASECELL(cell)
  if cell.buddies ⊆ free_cells[cell.level] then
    higher_cell = Merge(cell, cell.buddies)
    free_cells[cell.level].remove(cell.buddies)
    ReleaseCell(higher_cell)
  else
    free_cells[cell.level].add(cell)
```

Allocating Low-Priority Cells

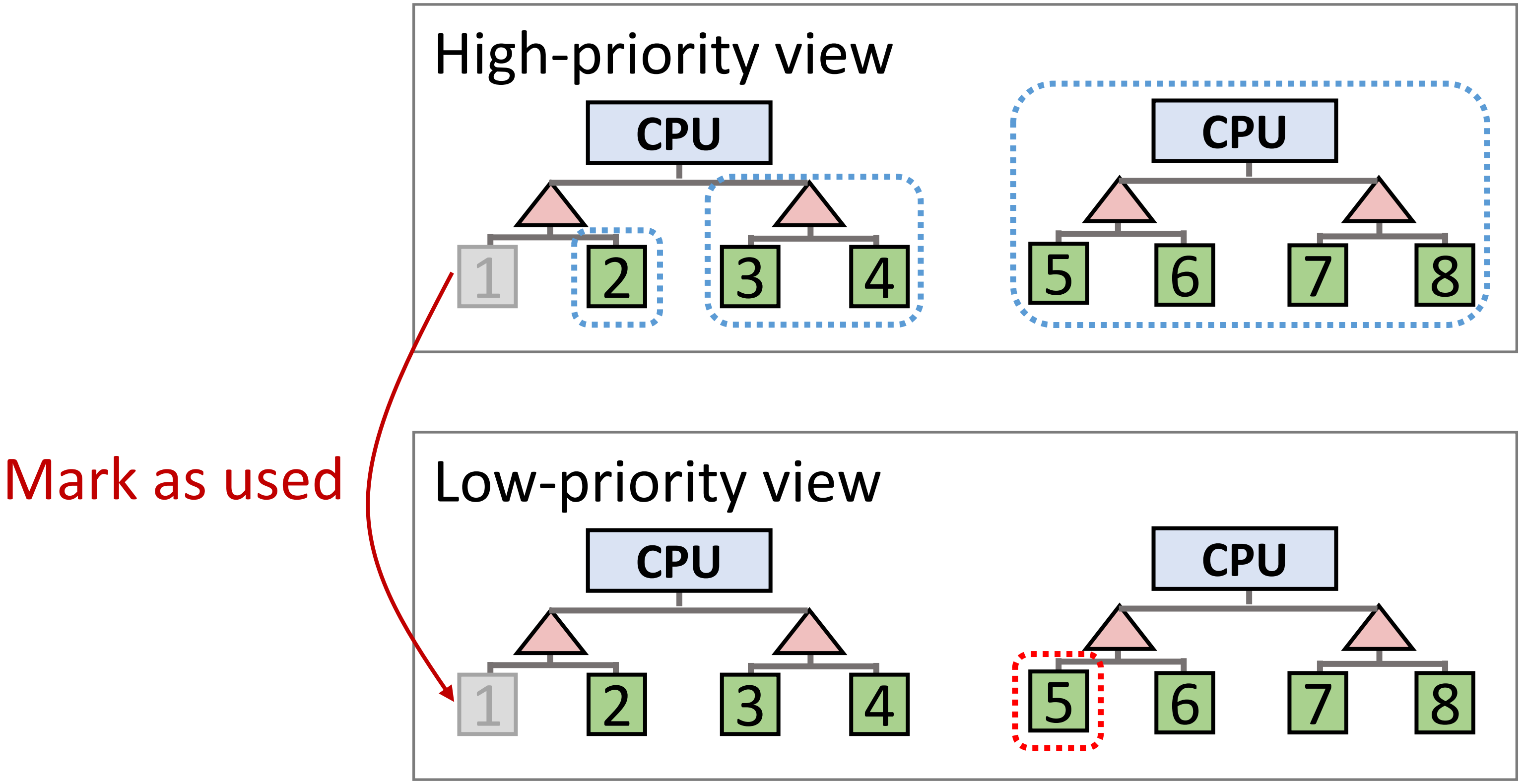
- Two cell views
  - High-priority guaranteed jobs with VC safety
  - Low-priority opportunistic jobs to improve utilization





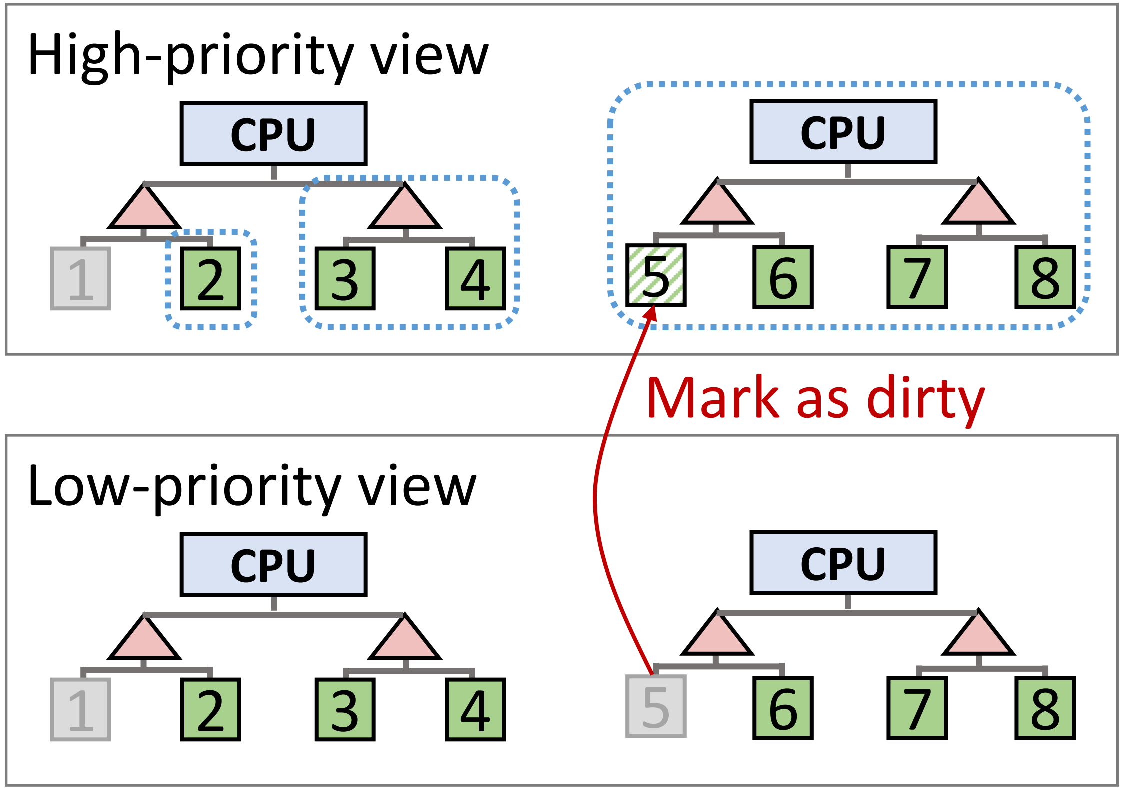
Allocating Low-Priority Cells

- Two cell views
  - High-priority guaranteed jobs with VC safety
  - Low-priority opportunistic jobs to improve utilization



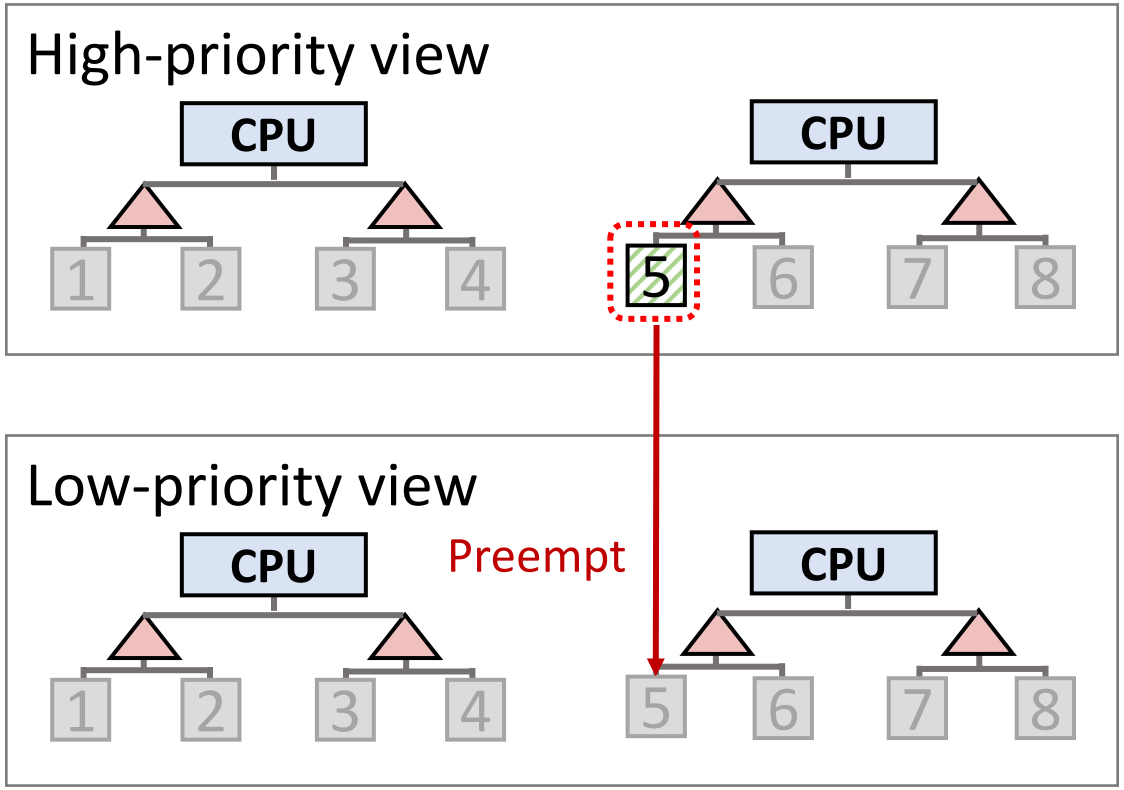
Allocating Low-Priority Cells

- Two cell views
  - High-priority guaranteed jobs with VC safety
  - Low-priority opportunistic jobs to improve utilization



Allocating Low-Priority Cells

- Two cell views
  - High-priority guaranteed jobs with VC safety
  - Low-priority opportunistic jobs to improve utilization



Open-Source Implementation

https://github.com/microsoft/hivedscheduler

- Implemented on Kubernetes

- Integrated with Microsoft OpenPAI

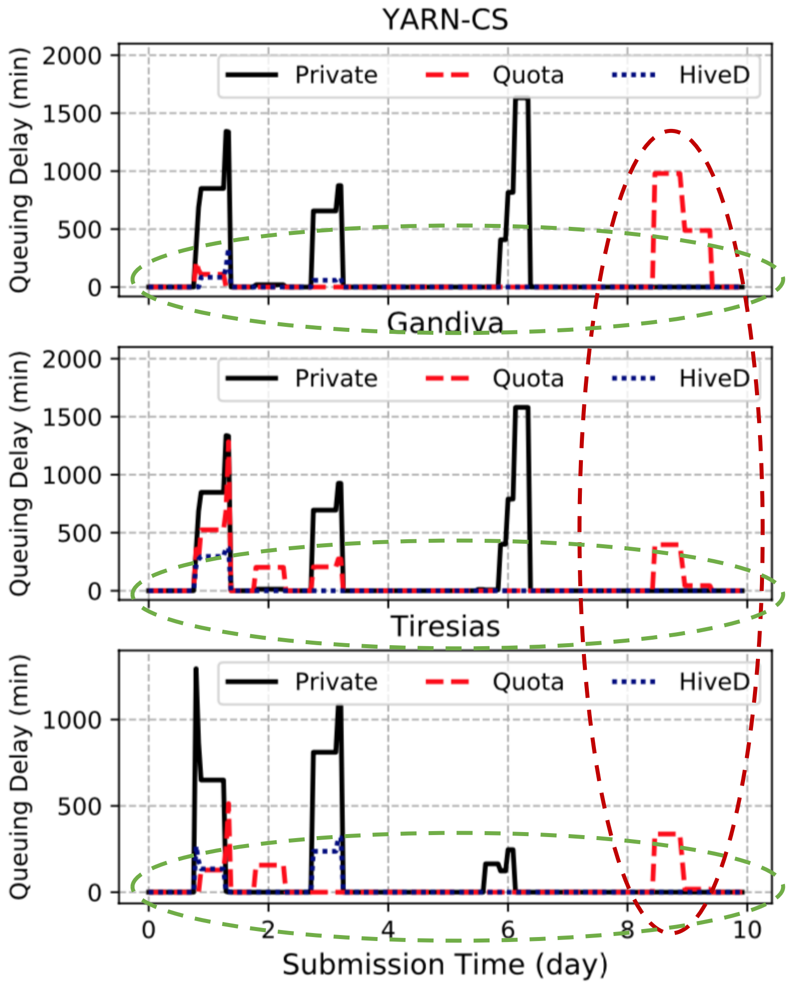
  https://github.com/microsoft/pai

- Deployed at Microsoft for 12+ months
  - Managing 1000+ heterogeneous GPUs
  - Serving research and production workloads at scale

- More implementation details and operation experiences in the paper

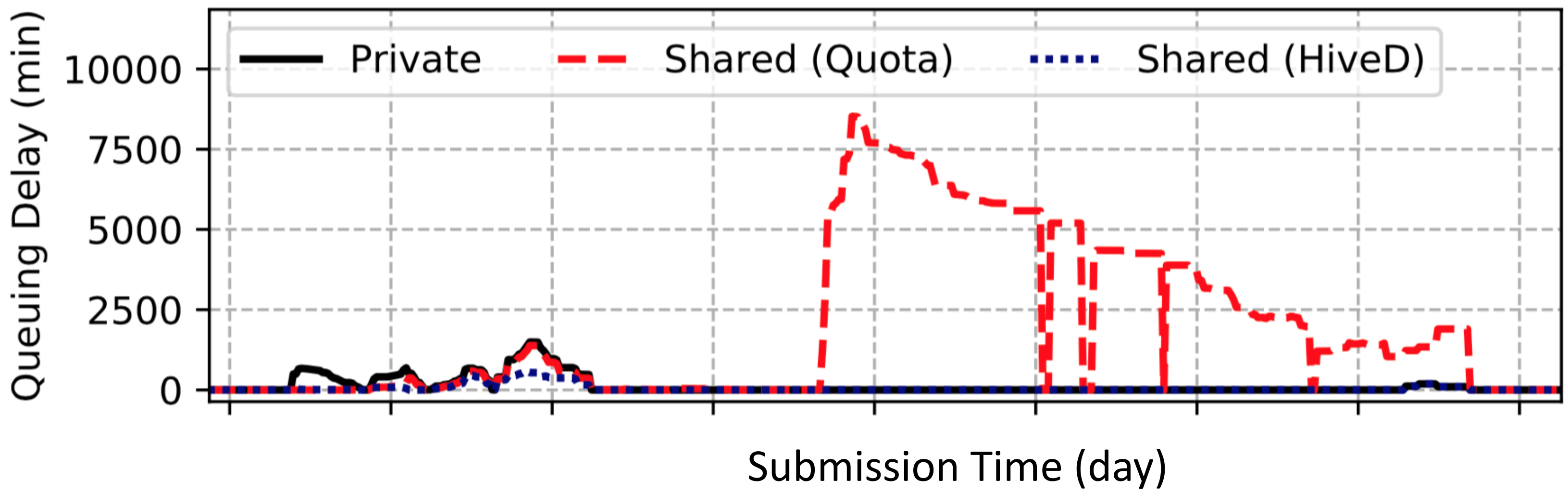
Evaluation: 96-GPU Cluster Experiment

- Schedulers
  - YARN-CS (Philly) [ATC 19]
  - Gandiva [OSDI 18]
  - Tiresias [NSDI 19]
  - A VC preserves the precise affinity structure, making STOA schedulers applicable
- HiveD achieves the best of both worlds
  - Sharing anomalies identified in all schedulers with quota; HiveD eliminates them all!
  - Significantly shorter queuing delay than in the private cluster
  - Similar overall JCT compared to applying the schedulers globally



Evaluation: Trace-Driven Simulation

- 2-month trace from a 2232-GPU cluster with 11 tenants
- More sharing anomalies under high load
  - Up to 8,000+ minutes of excessive queuing delay
  - 7x on average
  - Again HiveD eliminates them all

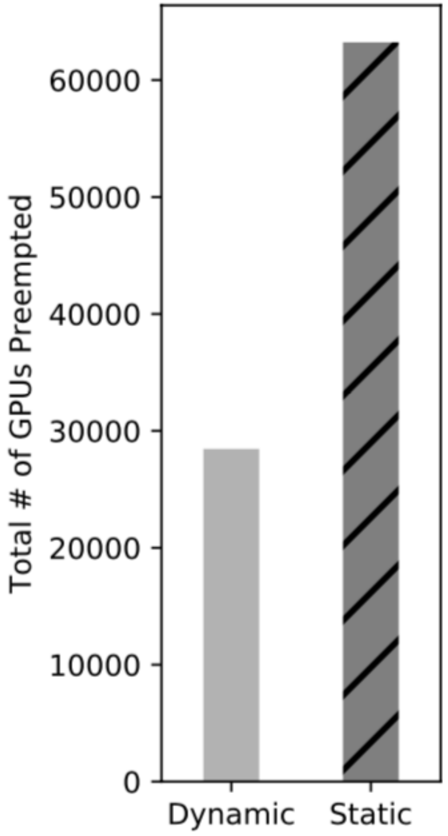


Evaluation: Trace-Driven Simulation

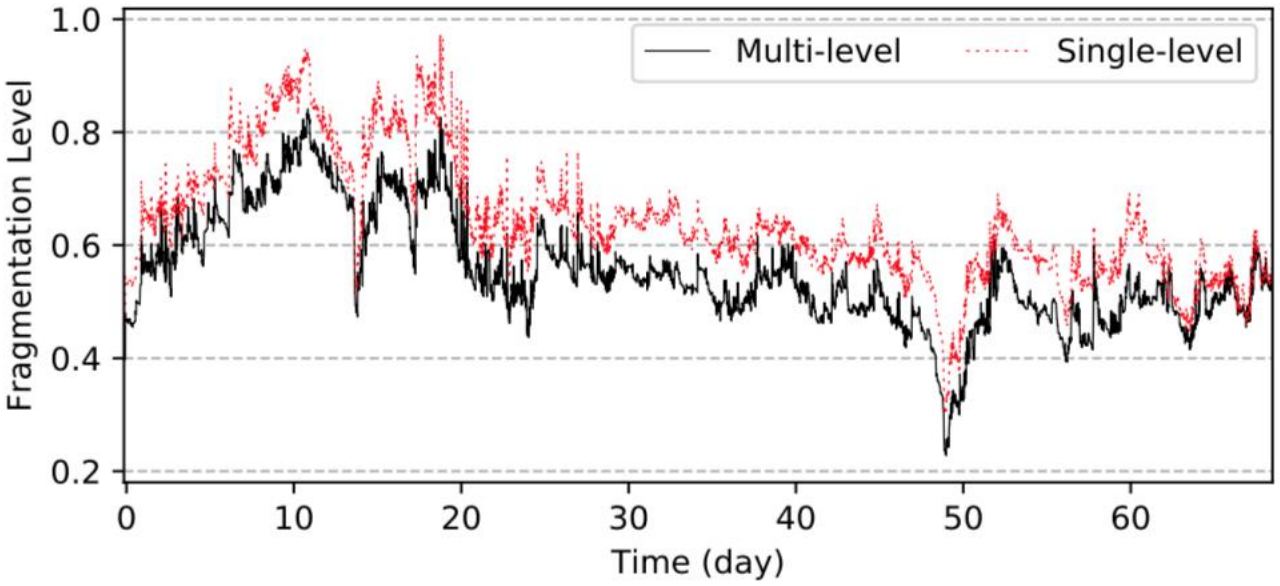
- 2-month trace from a 2232-GPU cluster with 11 tenants
- More sharing anomalies under high load
  - Up to 8,000+ minutes of excessive queuing delay
  - 7x on average
  - Again HiveD eliminates them all
- Sharing anomaly leads to diminishing benefits of sharing
  - Decommission a tenant with higher avg. queuing delay in the shared cluster with quota than in the private cluster
  - Tenants owning 37% quota (two large tenants) decommissioned!
  - Significantly longer queuing delay of the other 9 tenants in this smaller cluster

Evaluation: Buddy Cell Allocation

- Reducing preemptions
  - Avoid dirty cells with dynamic binding



- Reducing fragmentation
  - Pack cells across VCs



Conclusion

- HiveD addresses the challenge of sharing a GPU cluster with
  - *Sharing safety:* simple and practical guarantee easily appreciated by tenants
  - *Cell:* new resource abstraction for defining tenants' affinity structures
  - *Buddy cell allocation:* proven safety and support for low-priority jobs
  - *Two-layer architecture* to incorporate other scheduling goals while guaranteeing sharing safety

Thank you!

Contact

Hanyu Zhao (zhaohanyu1994@gmail.com)

Zhenhua Han (Zhenhua.Han@microsoft.com)

Code released at

https://github.com/microsoft/hivedscheduler