HiveD: Sharing a GPU Cluster for Deep Learning with Guarantees

Hanyu Zhao, *Peking University and Microsoft;* Zhenhua Han, *The University of Hong Kong and Microsoft;* Zhi Yang, *Peking University;* Quanlu Zhang, Fan Yang, Lidong Zhou, and Mao Yang, *Microsoft;* Francis C.M. Lau, *The University of Hong Kong;* Yuqi Wang, Yifan Xiong, and Bin Wang, *Microsoft*

https://www.usenix.org/conference/osdi20/presentation/zhao-hanyu

This paper is included in the Proceedings of the
14th USENIX Symposium on Operating Systems
Design and Implementation

November 4–6, 2020

978-1-939133-19-9

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*

Abstract

Deep learning training on a shared GPU cluster is becoming a common practice. However, we observe severe sharing anomaly in production multi-tenant clusters where jobs in some tenants experience worse queuing delay than they would have in a private cluster with their allocated shares of GPUs. This is because tenants use *quota*, the number of GPUs, to reserve resources, whereas deep learning jobs often use GPUs with a desirable *GPU affinity*, which quota cannot guarantee.

HiveD is the first framework to share a GPU cluster *safely*, so that such anomaly would never happen by design. In HiveD, each tenant reserves resources through a Virtual Private Cluster (VC), defined in terms of multi-level *cell* structures corresponding to different levels of GPU affinity in a cluster. This design allows HiveD to incorporate any existing schedulers within each VC to achieve their respective design goals while sharing the cluster safely.

HiveD develops an elegant *buddy cell allocation* algorithm to ensure *sharing safety* by efficiently managing the dynamic binding of cells from VCs to those in a physical cluster. A straightforward extension of buddy cell allocation can further support low-priority jobs to scavenge the unused GPU resources to improve cluster utilization.

With a combination of real deployment and trace-driven simulation, we show that: (i) sharing anomaly exists in three state-of-the-art deep learning schedulers, incurring extra queuing delay of up to 1,000 minutes; (ii) HiveD can incorporate these schedulers and eliminate the sharing anomaly in all of them, achieving separation of concerns that allows the schedulers to focus on their own scheduling goals without violating sharing safety.

1 Introduction

Deep learning training is becoming a major computing workload on a GPU cluster. It is a common practice for an organization to train deep learning models in a multi-tenant GPU cluster, where each tenant reserves resources using a quota that consists of the number of GPUs and other associated resources such as CPU and memory [52].

Surprisingly, in a production multi-tenant GPU cluster, we have observed unexpected anomalies where a tenant's deep learning training jobs wait significantly longer for GPUs than they would do in a private cluster whose size equals to the tenant's quota. This is because the current resource reservation mechanism is based on *quota*, i.e., the number of GPUs. Quota cannot capture the GPU *affinity* requirement of training jobs: e.g., an 8-GPU job on one node usually runs significantly faster than on eight nodes [41, 52, 86]. Quota cannot guarantee a tenant's GPU affinity like the tenant's private cluster does. As a result, multi-GPU jobs often have to wait in a queue or run at a relaxed affinity, both resulting in worse performance (longer queuing delay or slower training speed).

In this paper, we present HiveD, a resource reservation framework to share a GPU cluster for deep learning training that guarantees *sharing safety* by completely eliminating sharing anomalies. Instead of using quota, HiveD presents each tenant a virtual private cluster (abbreviated as VC) defined by a new abstraction: *cell*. Cell uses a multi-level structure to capture the different levels of affinity that a group of GPUs could satisfy. Those cell structures naturally form a hierarchy in a typical GPU cluster; e.g., from a single GPU, to GPUs attached to a PCIe switch, to GPUs connected to a CPU socket, to GPUs in a node, to GPUs in a rack, and so on.

With cell, HiveD virtualizes a physical GPU cluster as a VC for each tenant, where the VC preserves the necessary affinity structure in a physical cluster. This allows any state-of-the-art deep learning scheduler to make scheduling decisions within the boundary defined by the VC, without affecting the affinity requirement from other VCs, hence ensuring sharing safety. In this way, HiveD achieves the separation of concerns [47]: It focuses on the resource reservation mechanism and leaves other resource allocation goals to VC schedulers (e.g., cluster utilization and job completion time).

HiveD develops an elegant and efficient *buddy cell allocation* algorithm to bind cells from a VC to a physical cluster. Buddy cell allocation advocates dynamic cell binding over static binding for flexibility. It dynamically creates and releases the binding of cells in a VC to GPUs in the physical cluster, while providing *proven* sharing safety despite unpredictable workloads. Moreover, the algorithm can be naturally extended to support preemptible low-priority jobs to scavenge unused cells opportunistically to improve overall utilization. Combined, HiveD achieves the best of both a private cluster (for guaranteed availability of cells independent of other tenants) and a shared cluster (for improved utilization and access to more resources when other tenants are not using them).

We evaluate HiveD using experiments on a 96-GPU real

*Equal contribution.

cluster and trace-driven simulations. The evaluation shows that (i) sharing anomaly exists in all the evaluated state-of-the-art deep learning schedulers [41,52,86]; (ii) HiveD eliminates all sharing anomalies, decreases excessive queuing delay from 1,000 minutes to zero, while preserving these schedulers' design goals; (iii) HiveD guarantees sharing safety regardless of cluster loads, whereas a quota-based cluster can result in 7× excessive queuing delay for a tenant under a high load.

We have open-sourced HiveD [17], and integrated it in OpenPAI [20], a Kubernetes-based deep learning training platform. It has been deployed in multiple GPU clusters serving research and production workloads at scale, including a cluster of 800 GPUs where HiveD has been up and running reliably for more than 12 months (as of Nov. 2020).

In summary, this paper makes the following contributions:

- We are the first to observe and identify sharing anomaly in production multi-tenant GPU clusters for deep learning training.
- We define the notion of sharing safety against the anomaly and propose a new resource abstraction, i.e., multi-level cells, to model virtual private clusters.
- We develop an elegant and efficient buddy cell allocation algorithm to manage cells with proven sharing safety, and to support low-priority jobs.
- We perform extensive evaluations both on a real cluster and through simulation, driven by a production trace, to show that HiveD achieves the design goals in terms of sharing safety, queuing delay, and utilization.

2 Background and Motivation

The current approach of managing a multi-tenant GPU cluster. In large corporations, a large-scale GPU cluster is usually shared by multiple business teams, each being a tenant contributing their resources (budget or hardware). The tenants share the GPU cluster in a way similar to sharing a CPU cluster [1, 52]: Each tenant is assigned a number of tokens as its *quota*. Each token corresponds to the right to use a GPU along with other types of resource. The quota denotes an expectation that the tenant can access "at least" the share of resources it contributes.

To improve training speed in the cluster, a user usually specifies a GPU *affinity requirement* for a deep learning job [52, 86]. For example, it is often desirable for a 64-GPU job to run in the 8×8 affinity, i.e., to run the job on 8 nodes each with 8 GPUs, instead of 64×1, i.e., 64 nodes each using 1 GPU. Given the affinity requirements, the resource manager will satisfy them in a guaranteed (hard) or best-effort (soft) manner. If there is no placement satisfying a job's affinity requirement, the job will wait in the queue if it has a hard

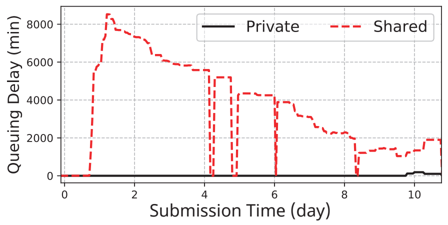


Figure 1: Sharing anomaly: a tenant suffers longer queuing delay in a shared cluster than in its own private cluster.

affinity requirement or will be scheduled with relaxed affinity if the requirement is soft (e.g., 64×1 as opposed to 8×8).

Sharing anomaly. In a production GPU cluster described in [52], we observe an anomaly from user complaints: a tenant is assigned a quota of 64 GPUs but reports that it cannot run a single (and the only) 8×8 deep learning job. Such anomaly arises because the tenant's assigned affinity has been fragmented, not by its own job(s) but by jobs from other tenants. Even though the tenant has enough GPU quota, the 64-GPU job has to wait in a queue or execute with degraded performance with relaxed affinity. The promise to the tenant that it can access at least its share of resource is broken.

Sharing anomaly appears similar to external fragmentation in memory management [54], if we liken a tenant to a program. The important difference however is that, in a shared GPU cluster, tenants expect their resource shares to be guaranteed. In the above real-world example, the fragmentation is due to other tenants, and the suffering tenant can hardly do anything except to complain to the cluster operator. Sharing anomaly can easily happen when jobs with lower affinity requirement (e.g., single-GPU jobs) from a tenant add to the fragmentation of global resources (due to varying job arrival and completion times), making jobs with higher affinity requirement (e.g., 8×8-GPU jobs) from other tenant(s) not able to run, even with sufficient quota. Apparently, quota can reserve only the quantity of resources, but not the affinity of resources. Hence it cannot automatically get around the external fragmentation across tenants. We call this phenomenon "sharing anomaly" because the sharing of a tenant's resource impacts the tenant negatively. Therefore, in the above case, rather than sharing with others, the wised up tenant would prefer to run a private cluster with eight 8-GPU nodes to adhere to its 8 × 8 GPU affinity with zero queuing delay.

A multi-tenant cluster is said to suffer from *sharing anomaly* if a tenant's sequence of GPU requests (possibly with affinity requirement) cannot be satisfied in this shared cluster; whereas it can be satisfied in a private cluster whose size equals to the tenant's quota. Figure 1 highlights how severe sharing anomaly could become, selected from a trace-driven simulation in a setup similar to [52] (more details in §5). The figure shows the job queuing anomaly of one tenant in a shared cluster when jobs have hard affinity requirement. In the 10-day submission window (denoted as X-axis), the tenant's average job queuing delay (denoted as Y-axis) in the shared cluster is significantly higher than that in its own private cluster.[1] In particular, the jobs submitted around Day 1 have to stay in the queue for more than 8,000 minutes (5 days) while they have zero queuing delay in the private cluster! Moreover, tenants having reserved large resources tend to suffer the most. Consequently, we have witnessed important corporate users reverting to private clusters, after experiencing high queuing delay brought by severe sharing anomalies.

One approach to reducing sharing anomaly is to devise a scheduling policy to minimize global resource fragmentation. This makes the design of a deep learning scheduler even more complex, which already has to manage sophisticated multi-objective optimizations. For example, minimizing global fragmentation may decrease job performance due to increased inter-job interference [86]. Therefore, we propose to separate the concern of sharing anomaly from other resource allocation objectives [47]. Instead of developing a complicated scheduler that achieves all possible goals, we design HiveD, a resource reservation framework that focuses on eliminating sharing anomaly, and provides a clean interface to incorporate any state-of-the-art deep learning schedulers to address concerns like cluster utilization [86], job completion time [41, 66], and fairness [29, 60].

3 HiveD Design

3.1 System Overview

HiveD proposes to guarantee *sharing safety* (i.e., eliminating sharing anomaly as described in §2) as a prerequisite of sharing a GPU cluster. Specifically, if a sequence of GPU requests with affinity requirements can be satisfied in a private cluster, it should be satisfied in the corresponding virtual private cluster and the shared physical cluster.

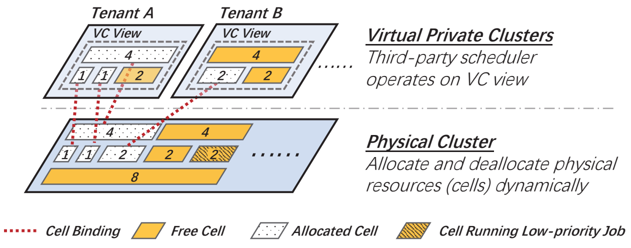

Tenant A
Tenant B
VC View
VC View
4
1
1
2
4
2
2
......
Virtual Private Clusters
Third-party scheduler operates on VC view
4
4
1
1
2
2
2
......
8
Physical Cluster
Allocate and deallocate physical resources (cells) dynamically
Cell Binding
Free Cell
Allocated Cell
Cell Running Low-priority Job

Figure 2: System architecture: a two-layer design.

Figure 2 illustrates the overall system architecture. HiveD's abstraction of GPU resources is divided into two layers, i.e., the layer of Virtual Private Clusters (VCs) and the layer of physical cluster. HiveD presents each tenant a VC. Each VC is pre-assigned a set of *cell*s, a novel resource abstraction that captures not only quota, but also the affinity structure of GPUs (the number inside each cell in the figure shows the number of affinitized GPUs of the cell). The cells assigned to a VC form a VC view with the GPU affinity structure identical to that of the corresponding private cluster. Any third-party scheduler can be incorporated to work on the VC view to achieve a certain goal of resource allocation [41, 52, 60, 86]. Moreover, HiveD ensures that any scheduling decision is constrained within the boundary defined by the VC view, as if happening on its private cluster, thus guaranteeing sharing safety.

Cells in a VC are logical. When a job uses a GPU in a logical cell, e.g., one GPU in the 4-GPU cell in the VC view of Tenant A in Figure 2, the logical cell will be bound to a physical cell allocated from the physical cluster, denoted at the bottom of Figure 2. If none of the GPUs is in use, the logical cell will be unbound from the physical cluster. To improve utilization, preemptible low-priority jobs can scavenge idle GPUs opportunistically. Such dynamic binding is more flexible than static binding: a dynamic binding can avoid a physical cell whose hardware is failing; it can avoid cells used by low-priority jobs to reduce preemptions; it can also pack the cells to minimize the fragmentation of GPU affinity.

To achieve this, HiveD adopts *buddy cell allocation*, an efficient and elegant algorithm, to handle the dynamic binding. A key challenge of dynamic binding is to guarantee the safety property in response to dynamic workloads, that is, jobs arrive unpredictably and request varying levels of cells. Buddy cell allocation algorithm is proven to ensure sharing safety: any legitimate cell request within a VC is guaranteed to be satisfied. The algorithm can also support low-priority jobs. Figure 2 shows a possible cell allocation, where cells in a physical cluster are bound to those defined in two VCs, and also to a low-priority job.

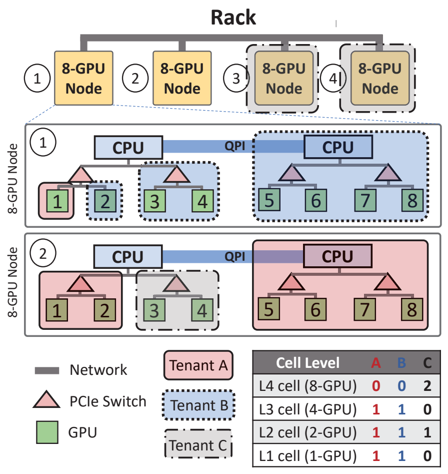
In §3.2, we explain the details of cells and show how a VC can be defined by cells. And in §3.3, we introduce the buddy cell allocation algorithm, prove its sharing safety guarantee, and extend it to support low-priority jobs.

3.2 Virtual Private Cluster with Cells

To model a (private) GPU cluster, HiveD defines a *hierarchy of multi-level cell structures*. A *cell* at a certain level is the corresponding collection of affinitized GPUs with their interconnection topology. Each virtual private cluster (VC) is then defined as number of cells at each level, modeled after the corresponding private cluster.

Figure 3 shows an example, where there are 4 levels of cell structures: at the GPU (level-1), PCIe switch (level-2), CPU socket (level-3), and node levels (level-4), respectively. The cluster has one rack that consists of four 8-GPU nodes, shared by three tenants, *A*, *B*, and *C*. The cell assignment for each tenant's VC is summarized in the table in Figure 3. Tenants *A* and *B*'s VCs both reserve one level-3 cell (4 GPUs under the same CPU socket), one level-2 cell (2 GPUs under the same

[1]The anomaly is dominated by queuing delay in the job completion time when the affinity requirement is hard. Details discussed in §5.1.



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

Figure 3: Multi-level cell assignment for a rack: an example.

PCIe switch), and one level-1 cell (single GPU). Tenant *C* is a larger tenant, which reserves two level-4 cells (node level) and one level-2 cell. Given the VC views defined in Figure 3, HiveD can adopt a third-party scheduler [41, 52, 60, 86] to work on the views. From the third-party scheduler's point of view, the VC view is no different from a private cluster consisting of nodes with different sizes (i.e., different level of cells). For example, the scheduler can treat tenant *C* as a private cluster with two 8-GPU nodes and one 2-GPU node, despite the fact that the 2-GPU node is actually a level-2 cell. Note that a third-party scheduler can use any GPUs in the assigned cells. For example, it can schedule two 2-GPU jobs to a 4-GPU (level-3) cell: a cell is the granularity of resource reservation in VCs and the physical cluster, but not necesarily the job scheduling granularity of a third-party scheduler.

In the cell hierarchy, a level-k cell c consists of a set S of level-$(k-1)$ cells. The cells in S are called *buddy cells*; buddy cells can be merged into a cell at the next higher level. We assume cell demonstrates *hierarchical uniform composability*: (i) all level-k cells are equivalent in terms of satisfying a tenant request for a level-k cell, and (ii) all level-k cells can be split into the same number of level-$(k-1)$ cells.

Heterogeneity. A heterogeneous cluster can be divided into multiple homogeneous ones satisfying hierarchical uniform composability. This is logical in practice because a production cluster typically consists of sufficiently large homogeneous sub-clusters (each often a result of adding a new GPU model and/or interconnect) [52]. Users typically use homogeneous GPUs for a job for better performance and specify the desired GPU/topology type (e.g., V100 vs. K80).

Initial cell assignment. A cluster provider must figure out the number of cells at each level to be assigned to each tenant's VC. A VC assignment is *feasible* in a physical cluster if it can accommodate all cells assigned to all VCs; that is, there exists a one-to-one mapping from the logical cells in each VC to the physical cells in the physical cluster. The initial cell assignment for VCs depends on factors like budget, business priority, and workload, thus it is handled outside of HiveD (§6 for further discussion). A cluster might spare more physical resources than the assigned cells to handle hardware failures.

Algorithm 1 Buddy Cell Allocation Algorithm

```
1: // Initial state of free_cells: only top level has cells
2: procedure ALLOCATECELL(cell_level)
3:   if free_cells[cell_level].size() == 0 then
4:     c = AllocateCell(cell_level+1)
5:     cells = Split(c)                    ▷ Split cells are buddies
6:     free_cells[cell_level].extend(cells)
7:   Return free_cells[cell_level].pop()
8:
9: procedure RELEASECELL(cell)
10:  if cell.buddies ⊆ free_cells[cell.level] then
11:    higher_cell = Merge(cell, cell.buddies)
12:    free_cells[cell.level].remove(cell.buddies)
13:    ReleaseCell(higher_cell)
14:  else
15:    free_cells[cell.level].add(cell)
```

Note that dashed lines in Figure 3 illustrate only one possible cell binding. HiveD advocates dynamic cell binding for flexibility, which reduces job preemption and fragmentation of GPU affinity. §5.3 confirms its benefits over static binding.

3.3 Buddy Cell Allocation Algorithm

HiveD manages the dynamic binding between the logical cells in VCs and the physical cells in the physical cluster, and handles requests to allocate and release cells. This is done by the *buddy cell allocation* algorithm. The algorithm maintains for each VC the information of (i) the corresponding physical cell for each allocated logical cell (i.e., the binding); (ii) a global free list at each cell level k to track all unallocated physical cells at that level. The algorithm always keeps available cells at the highest possible level: for example, if all the buddy cells at level-$(k-1)$ are available for a cell at level-k, only the cell at level-k is recorded. And the algorithm aims to keep as many higher-level cells available as possible. Algorithm 1 shows the pseudo-code of the algorithm.

To allocate a level-k cell in a VC, the algorithm starts at level-k and goes up the levels if needed: it first checks whether a free level-k cell is available and allocates one if available. If not, the algorithm will move up level-by-level, until a free level-l cell is available, where $l > k$. The algorithm will then split a free level-l cell recursively into multiple lower-level cells, until a level-k cell is available. Each splitting produces a set of buddy cells at the next lower level, which will be added to the free list at that lower level. One of those new low-level cells is again split until free level-k cells are produced.

The cell release process also works in a bottom-up manner. When a level-k cell c is released, the algorithm adds c into the free list of level-k cells and checks the status of c's buddy cells. If all of c's buddy cells are free, the algorithm will merge c and its buddy cells into a level-$(k+1)$ cell. The merge process continues recursively while going up the levels, until no cells can be merged. In this way, the buddy cell allocation algorithm reduces GPU fragmentation and creates opportunities to schedule jobs that require higher-level cells.

Before processing an allocation request, the algorithm ensures the request is *legal* in that it is within the assigned quota for the VC at this cell level. HiveD stores the cell assignment in a table r, where a tenant t's preassigned number for level-k cells is stored in $r_{t,k}$. The buddy cell allocation algorithm guarantees to satisfy all legal cell requests under a feasible initial VC assignment, which is formally stated in Theorem 1.

Theorem 1. *Buddy cell allocation algorithm satisfies any legal cell allocation, under the condition of hierarchical uniform composability, if the original VC assignment is feasible.*

Proof. Denote as $r_{t,k}$ the number of level-k cells reserved by tenant t, i.e., cell assignment for t. Denote as r_k the number of reserved level-k cells for all tenants, i.e. $r_k = \sum_t r_{t,k}$. Denote as $a_{t,k}$ the number of level-k cells that have already been allocated to t by the buddy cell allocation algorithm. Cell allocations that maintain $a_{t,k} \leq r_{t,k}$ are legal. Denote as a_k the number of allocated level-k cells for all tenants (i.e., $a_k = \sum_t a_{t,k}$), and f_k the number of free level-k cells in the physical cluster, and h_k the number of level-$(k-1)$ buddy cells that a level-k cell can be split into (hierarchical uniform composability). Define F_k as the number of level-k cells that can be obtained by splitting the higher level cells while still satisfying the safety check for the cell assignment. F_k can be calculated by Eqn. (1).

$$F_k = \begin{cases} (f_{k+1} + F_{k+1} - (r_{k+1} - a_{k+1}))h_{k+1} & k < \hat{k}; \\ 0 & k = \hat{k}, \end{cases} \tag{1}$$

where $\hat{k}$ is the highest level.

To prove the theorem, we prove the following invariant:

$$r_k - a_k \leq f_k + F_k \qquad \forall k = 1, 2, ..., \hat{k}. \tag{2}$$

The L.H.S. is the number of level-k cells all tenants have yet to allocate, and the R.H.S. is the number of available level-k cells the cluster can provide.

We prove by induction on discrete time slots. Denote as w the sequence number of time slots. A change of the cluster state will increase w by 1. When $w = 0$, $a_k = 0$, the invariant (2) holds as long as the original VC assignment is feasible. Assuming the invariant holds at time $w = i$, we shall prove the invariant still holds at time $w = i+1$ after a tenant allocates a legal level-k cell.

Because the allocation is legal, $a_k < r_k$ should hold at time $i+1$. In order to satisfy the invariant (2), either $f_k > 0$ or $f_k = 0$.

When $f_k > 0$, according to Algorithm 1, $a_k = a_k + 1$ and $f_k = f_k - 1$ after an allocation of level-k cell at time $i+1$. The gap of both sides in the invariant remains constant, thus it still holds.

When $f_k = 0$, i.e., no free cell at level-k, the algorithm will split a level-k' cell by finding the smallest k' where $k' > k$ and $f_{k'} > 0$. In this case, the invariant remains true as in the $f_k > 0$ case, while the gap of the invariant at level-k' will decrease by 1. If the invariant at the level-k' breaks after cell splitting, it would mean $r_{k'} - a_{k'} = f_{k'} + F_{k'}$ at time $w = i$. By definition, F_k should be 0 at time $w = i$. But since $a_k < r_k$ (because the allocation request is legal), thus the invariant (2) cannot hold true at level k. This leads to a contradiction. Therefore, the invariant must hold at level k' after splitting a level-k' cell. Following the same step, we can prove the invariant holds at level k'' when the algorithm recursively splitting a level-k'' cell, where $k'' \in [k+1, k'-1]$. Hence the invariant holds on all levels when $f_k = 0$.

Merging the buddy cells can only either increase or keep the gap of the invariant and thus it still holds. Q.E.D. □

The buddy cell allocation algorithm has the time complexity of $O(\hat{k})$, where $\hat{k}$ is the number of levels, and can therefore scale to a large GPU cluster efficiently: $\hat{k}$ is usually 5, from the level of racks to the level of GPUs.

Hierarchical uniform composability ensures the algorithm's correctness and efficiency: it does not have to check explicitly after each split whether or not the subsequent legal allocation requests are satisfiable. Instead, it just needs to check whether every allocation request is legal. For the case where cells are heterogeneous (e.g., due to different GPU models or different inter-GPU connectivities), HiveD partitions the cluster into several pools within which cells at the same level are homogeneous, and applies Algorithm 1 in each pool.

The algorithm resembles buddy memory allocation [56], hence the name. Beyond reducing fragmentation efficiently [35], our key contribution here is making the non-obvious observation: GPU affinity can be modeled as cells, thus making buddy allocation applicable. Moreover, we prove that buddy cell allocation satisfies sharing safety, while traditional buddy allocation does not have such safety concern and hence does not provide this guarantee. Our algorithm also reveals the different characteristics of GPU hierarchy vs. memory regions; for example, the hierarchical uniform composability condition captures GPU hierarchy and is a generalization of the artificially-created power-of-2 rule in buddy memory allocation. Our algorithm also supports priority (elaborated next).

Allocating low-priority cells. The buddy cell allocation algorithm can be naturally extended to support low-priority jobs (a.k.a. opportunistic jobs), whose allocated cells can be preempted by high-priority jobs. Supporting such low-priority jobs helps improve overall GPU utilization, without compromising the sharing safety guarantees provided to the

VCs. HiveD maintains two cell views, one for allocating high-priority (guaranteed) cells, and the other for allocating the low-priority cells. Both views manage the same set of cells in the physical cluster using the same cell allocation algorithm (i.e., Algorithm 1). Similar to YARN [83] and Omega [73], HiveD enforces strict priority where high-priority bindings can preempt low-priority cells. Note that preempting a low-priority job could lead to loss of training progress if its checkpoint is stale. When allocating low-priority cells, HiveD chooses the cells farthest away from those occupied by high-priority jobs (e.g., a non-buddy cell of a high-priority cell) in order to minimize the chance of being preempted. Likewise, when allocating high-priority cells, HiveD chooses the free cells with the fewest GPUs used by low-priority jobs to reduce the chances of unnecessary preemptions. With a similar approach, we can extend HiveD to support multiple levels of priority.

HiveD adopts weighted max-min fairness [37,49] to decide the numbers of low-priority cells allocated to tenants. One could incorporate other state-of-the-art fairness metrics [60] to decide the fair share among tenants.

4 Implementation

HiveD has been integrated in OpenPAI, an open-source deep learning training platform [20] based on Kubernetes [28]. It has been deployed to multiple GPU clusters, managing various types of GPUs from NVIDIA Volta [19] to AMD MI50 [14]. This includes a cloud cluster with 800 heterogeneous GPUs (200 Azure GPU VMs) where HiveD has been running reliably for 12+ months (as of Nov. 2020). HiveD has served research and production workloads at scale, ranging from long-lasting training of large NLP models (e.g., BERT large [34]) to AutoML experiments that consist of hundreds of short-lived 1-GPU jobs. Next we share our experience in implementing and operating HiveD.

HiveD is implemented in 7,700+ lines of Go codes. In addition, it has a few more thousands of lines of JavaScript, Shell scripts, and YAML specifications to integrate with the training platform. It is implemented as a *scheduler extender* [9], a standalone process that works in tandem with the Kubernetes default scheduler (kube-scheduler [7]). This way, HiveD is able to reuse kube-scheduler's basic scheduling logic.

Cell specification. HiveD relies on a cell specification to understand the cell hierarchies in a cluster and the cell assignments for VCs. Figure 4 presents an example specification for a heterogeneous GPU cluster with two racks of NVIDIA V100 GPUs and one rack of NVIDIA P100 GPUs. `cellHierarchy` describes the two types of multi-level cell structures. `physicalCluster` specifies the cell layout in a physical cluster: two V100 racks and one P100 rack, and their IP addresses. With `physicalCluster` and `cellHierarchy`, `vcAssignment` specifies the cell assignment for a VC: the only P100 rack and 4 V100 nodes are assigned to the VC `vc1`.

```
cellHierarchy:
- name: V100-RACK # cell hierarchy for V100 rack
  hierarchy:
  - cellType: V100-GPU  # level-1 cell
  - cellType: V100-PCIe-SWITCH
    splitFactor: 2 # split to 2 level-1 cells
  - cellType: V100-CPU-SOCKET
    splitFactor: 2
  - cellType: V100-NODE
    splitFactor: 2
  - cellType: V100-RACK # level-5 (top-level)cell
    splitFactor: 8
- name: P100-RACK # cell hierarchy for P100 rack
  hierarchy:
  - cellType: P100-RACK # omit lower-level cells
    splitFactor: 8

physicalCluster:                vcAssignment:
- topLevelCellType: V100-RACK   - vc: vc1 #omit other VCs
  topLevelCellAddresses:          cells:
  - 10.0.1.0~7                    - subCluster: P100-RACK
  - 10.0.2.0~8                      - cellType: P100-RACK
- topLevelCellType: P100-RACK         cellNumber: 1
  topLevelCellAddresses:          - subCluster: V100-RACK
  - 10.0.3.0~7                      - cellType: V100-NODE
                                      cellNumber: 4
```

Figure 4: A simplified cell specification (in `.yaml` format).

A third-party scheduler can leverage the VC view of `vc1` to make scheduling decisions, as if `vc1` is a physical cluster. Our release of HiveD comes with a tool to automatically detect infeasible VC assignments in the specification.

Handling faulty hardware. When multiple free cells are available, the buddy cell allocation algorithm allows HiveD to avoid using faulty hardware. It prefers binding to a healthy cell when possible. When a VC has no other choice, HiveD will proactively bind to a faulty physical cell so that the third-party scheduler in the VC can see the faulty hardware and avoid using GPUs in the cell.

Fault tolerance. The HiveD process itself is also fault-tolerant. It is deployed as a Kubernetes StatefulSet [10] to ensure a single running instance. HiveD maintains several centralized in-memory data structures to keep all the run-time information used for cell allocation (e.g., the free cell list, and the cell allocation list). To reduce overheads, these data structures are not persistent. HiveD partitions and stores the cell binding decision for each pod in its "pod annotation", which is kept reliably by Kubernetes. If a job has multiple pods, the annotation in each pod stores the cell binding decisions for all the pods of the job. When recovering from a crash, HiveD reconstructs all the in-memory data structures like the cell allocation list and the free cell list from the pod annotation in all the running pods. Moreover, with the cell binding decisions stored in pod annotation, HiveD could detect whether or not there are unscheduled pods and resume the scheduling for the unscheduled ones. In case none of the pods of a job gets scheduled when HiveD crashes, the job manager, another single instance StatefulSet, will receive a timeout and resubmit the job. The fault tolerance of the third-party scheduler is handled by the scheduler itself.

Reconfiguration. We observe that a cluster operator may occasionally change the cell specification on-the-fly to reconfigure a cluster: adding, removing, or upgrading hardware; adjusting cell assignment for a VC. HiveD treats reconfiguration similar to crash recovery. The difference is during a reconfiguration HiveD will check if there is any inconsistency between the old cell bindings in the pod annotations and the new cell specification. For example, the total bound cells from a VC may exceed the new cell assignment. In this case, HiveD will downgrade the jobs with the inconsistent pods to low-priority jobs and preempt them when necessary.

The failure handling and reconfiguration capabilities of HiveD have been tested and verfied on all the deployed OpenPAI clusters. There are occasional hardware issues that require human intervention, e.g., power failures, GPU hardware failures. HiveD handles the decommission and recommission of hardware smoothly. To fully validate its failure handling capability, we run HiveD on an 800-GPU cluster on 200 Azure low-priority VMs [78]. The 200 Azure VMs consist of 125 NC24 [15] (NVIDIA Tesla K80) and 75 NV24 [16] (NVIDIA Tesla M60) series VMs, which could get preempted anytime. HiveD treats a preempted VM as a faulty cell. When a preempted VM resumes, HiveD will re-include it in the cluster just as a faulty cell turning normal. We observe up to 75% of the preemption rate (150 out of 200 VMs) in the cluster. And HiveD handles the preemptions well. When a VM gets preempted, the deep learning job running atop will migrate to other available GPUs or wait in a queue when GPUs are unavailable. The waiting job will get scheduled within one minute when a desired VM resumes from preemption.

5 Evaluation

We evaluate HiveD using experiments on a 96-GPU cluster on a public cloud and trace-driven simulations on a production workload. Overall, our key findings include:

- HiveD eliminates all the sharing anomalies found in all the tested schedulers. Excessive job queuing delay decreases from 1,000 minutes to zero.
- HiveD can incorporate the state-of-the-art deep learning schedulers and complement them with sharing safety, while maintaining their scheduling goals and preserving sharing benefits with low-priority jobs.
- HiveD guarantees sharing safety under various cluster load. In contrast, high cluster load in quota-based scheme can result in 7× excessive queuing delay.
- HiveD's buddy cell allocation algorithm reduces job preemption by 55% with dynamic binding and fragmentation of GPU affinity by up to 20%.

Experimental setup. We collect a 2-month trace from a production cluster of 279 8-GPU nodes (2,232 GPUs). The trace contains 141,950 deep learning training jobs, each specifying its submission time, training time, number of GPUs with the affinity requirement, and the associated tenant. The cluster is shared by 11 tenants. Table 1 shows each tenant's quota assignment in the real deployment and the distribution of a job's GPU number. Please refer to [52] for more details of the trace and its collection and analysis methodology. We run experiments in a 96-GPU cluster deployed on Azure. The cluster consists of 24 virtual machines (NC24 [15]), each with 4 NVIDIA K80 GPUs.

Tenant	1-GPU	2-GPU	4-GPU	8-GPU	≥16-GPU	Total	Quota
res-a	429	14	260	625	40	1,368	0.37%
res-b	18,319	1,593	931	148	238	21,229	0.73%
res-c	3,285	161	716	185	0	4,347	0.73%
res-d	1,754	0	0	0	0	1,754	1.47%
res-e	2,682	110	3,005	0	0	5,797	1.83%
res-f	8,181	88	618	1,337	559	10,783	28.57%
prod-a	227	54	23	1,132	138	1,574	8.79%
prod-b	16,446	67	605	1,344	22	18,484	10.62%
prod-c	4,692	301	1,905	4,415	1,206	12,519	11.36%
prod-d	781	6	545	650	95	2,077	15.75%
prod-e	58,407	532	2,118	959	2	62,018	19.78%
Total	115,203	2,926	10,726	10,795	2,300	141,950	100%

Table 1: Number of jobs with different GPU demands and quota assignment of tenants.

5.1 Sharing Safety: Cluster Experiments

In this section, we examine sharing safety in traditional quota-based scheme and HiveD on the deployed cluster.

Methodology. We collect a 10-day trace from the original 2-month production trace. To approximate the load of the 2,232-GPU cluster on a 96-GPU one, we scale down the number of jobs by randomly sampling from the 10-day trace proportionally (96 out of 2,232). Due to security reasons, we do not have access to the code and data of the jobs. Therefore, we replace the jobs with 11 popular deep learning models in domains of Natural Language Processing (NLP), Speech, and Computer Vision (CV) from GitHub (summarized in Table 2). We mix these models following a distribution of NLP:Speech:CV = 6:3:1, as reported in [86].

We test three state-of-the-art deep learning schedulers: YARN-CS [52], Gandiva [86], and Tiresias [41]. We obtained the source code of Gandiva and Tiresias [11], and use the same implementation in our experiments. YARN-CS is a modified YARN Capacity Scheduler. It packs jobs as close as possible to spare good GPU affinity, similar to [52]. We further refine the preemption policy of YARN-CS: instead of preempting the latest jobs, it preempts low-priority jobs based on the desired GPU affinity requirement. Otherwise, the baseline of YARN-CS will be much worse. To enforce quota in Tiresias, jobs exceeding the quota will get scheduled in a low-priority queue, which is also sorted by Tiresias. For each scheduler, we compare: (i) each tenant running its jobs in a private cluster with the capacity set to its quota; (ii) tenants sharing the

Type	Model	Dataset
NLP	Bi-Att-Flow [74]	SQuAD [68]
	Language Model [90]	PTB [61]
	GNMT [85]	WMT16 [13]
	Transformer [82]	WMT16
Speech	WaveNet [81]	VCTK [12]
	DeepSpeech [45]	CommonVoice [5]
CV	InceptionV3 [79]	ImageNet [33]
	ResNet-50 [46]	ImageNet
	AlexNet [57]	ImageNet
	VGG16 [77]	ImageNet
	VGG19	ImageNet

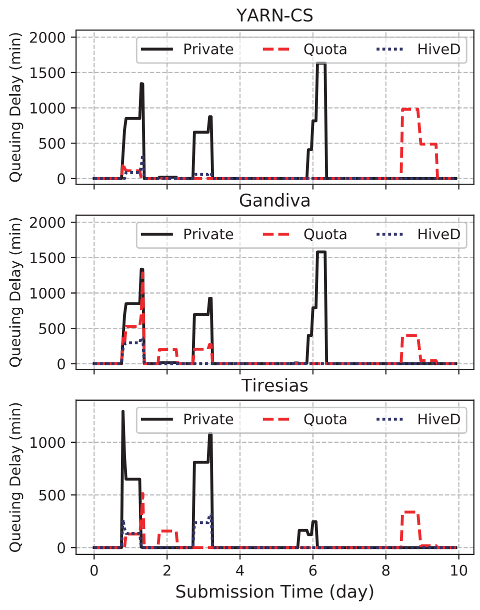
Table 2: Deep learning models used in the experiments [86].

cluster using quota; and (iii) tenants sharing the cluster using the scheduler with HiveD enabled. In a shared cluster, all schedulers will schedule jobs as high-priority ones if the tenant has sufficient resources in its quota or VC, otherwise the job will be scheduled as a low-priority one.

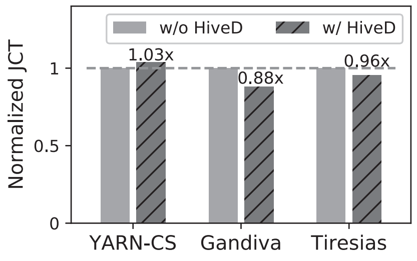
In HiveD's experiments, we use a cell hierarchy with four levels: node (8-GPU), CPU socket (4-GPU), PCIe switch (2-GPU), and GPU. We assign each tenant a set of node-level cells with a total number of GPUs equal to its quota. To model the cell hierarchy after the production cluster, we treat every two contiguous 4-GPU VMs as one logical 8-GPU node (i.e., one 8-GPU node level cell). Similar to [86], to speed up replaying the 10-day trace, we "*fast-forward*" the experiment by instructing running jobs to skip a number of iterations whenever there are no scheduling events, including job arrival, completion, preemption, migration, etc. The time skipped is calculated by measuring job training performance in a stable state. To enable the skipping, HiveD bypasses the kube-scheduler and talks to job pods directly.

The trace shows that the GPU affinity requirements of most jobs are hard, showing that users are not willing to sacrifice training performance. In this case, queuing delay is the major source of sharing anomaly in the overall job completion time (JCT). Note that JCT consists of queuing delay and actual training time, and job training time is highly deterministic as long as GPU affinity is the same [86]. Therefore, we show the queuing delay to illustrate the sharing anomaly when job's GPU affinity requirement is hard. We also evaluate the JCT when job's affinity requirement is soft.

Results. Figure 5(a) shows the queuing delay of jobs from tenant prod-a using the three schedulers. The X-axis denotes the job submission time. The Y-axis denotes the queuing delay averaged in a 12-hour moving window. Figure 5(a) shows that all the three schedulers demonstrate sharing anomaly without HiveD. For YARN-CS, from Day 8 to Day 10, jobs in prod-a suffer 1,000 minutes longer queuing delay in a quota-based cluster than in its private cluster. Although YARN-CS packs jobs as compactly as possible, a large number of 1-GPU jobs from other tenants with varying durations make the available GPUs affinity highly fragmented. As a result, multi-GPU jobs have to wait a long time for the desired affinity. Since the



(a) Average queuing delay of Tenant prod-a



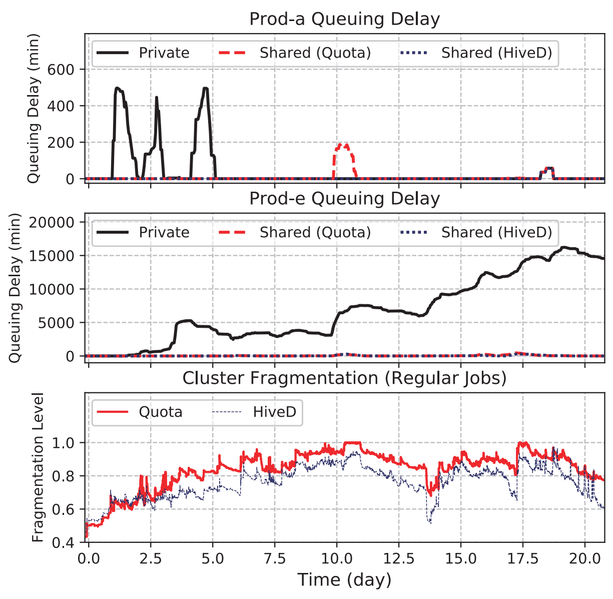
(b) Average job completion time across all tenants

Figure 5: The experiments for the three schedulers in a 96-GPU cluster, with and without HiveD.

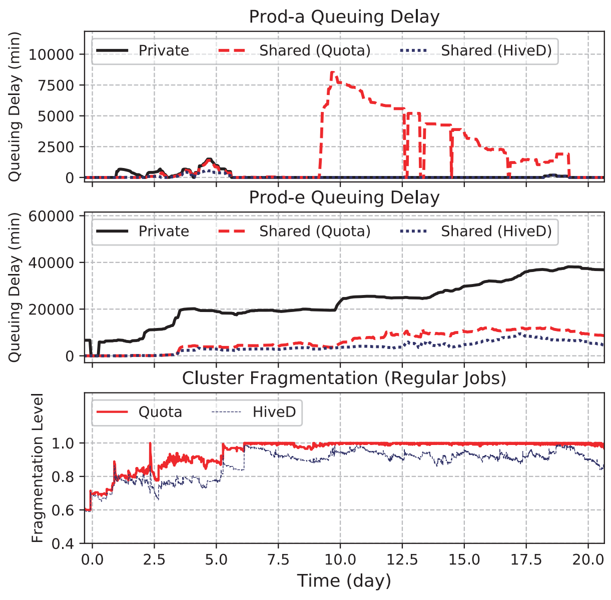
majority of jobs in prod-a use multiple GPUs (Table 1), the tenant suffers more from sharing anomaly.

Similarly, in Gandiva, jobs in prod-a suffer up to 400 minutes longer queuing delay in the shared cluster on Day 2 and Day 8. The excessive queuing delay is shorter than that in YARN-CS because Gandiva can mitigate the fragmentation of GPU affinity by job migration. However, unaware of cells in a VC, Gandiva's greedy algorithm may accidentally migrate jobs to improve the job performance in a tenant at the expense of other tenant's GPU affinity, thus violating safety. For example, Gandiva may greedily migrate away an interfering job in a VC while increasing the fragmentation and violating the sharing safety of other VCs. In contrast, HiveD achieves separation of concerns, allowing Gandiva to migrate jobs for its own goal without worrying about sharing safety. We will discuss job migration more in §6.

In Tiresias, Tenant prod-a shows sharing anomaly on Day 2 and Day 8. With quota enforcement, Tiresias suffers over 330 minutes longer queuing delay than that in its private cluster. To reduce job completion time (JCT), Tiresias prefers running



(a) 279 nodes



(b) 200 nodes

Figure 6: The average job queuing delay of Tenant prod-a and prod-e vs. the level of fragmentation of GPU affinity.

shorter and smaller jobs first. We do observe shorter queuing delay (and JCT) in Tiresias, compared to the other two schedulers. However, without HiveD, the global advantage of small jobs in a tenant might increase the fragmentation of GPU affinity in other tenants, thus resulting in sharing anomaly.

The experiment suggests that the evaluated schedulers are effective in their design objectives but they do not consider sharing safety, a factor that could severely impact user experience. HiveD complements the three schedulers with sharing safety by reserving the GPU affinity in each tenant's VC. With HiveD, prod-a (and all the other tenants) never experiences an excessive queuing delay in the shared cluster, using each of the three schedulers. Even during Days 8∼10, the multi-GPU jobs are scheduled immediately as the tenant has enough 8-GPU cells in its VC (hence the reserved cells in the physical cluster). HiveD also allows jobs to have a significantly shorter queuing delay in the shared cluster when a tenant runs out of its own capacity in the private cluster (Days 1, 3, and 6), by giving it chances to run low-priority jobs.

With sharing safety, HiveD can still preserve the scheduling efficiency. Figure 5(b) shows that HiveD exhibits similar job completion time compared to those without HiveD: at most 3% worse (for YARN-CS) and 12% better (for Gandiva).

We also evaluate the job completion time (JCT) when job's GPU affinity requirement is soft. Without HiveD, some jobs experience worse training speed due to a relaxed affinity requirement and thus result in higher JCT in a shared cluster than in a private cluster (i.e., sharing anomaly). Again, HiveD eliminates all sharing anomalies in this case. Overall we observe a trend similar to the result when the affinity requirement is hard, hence the details are omitted in this paper.

5.2 Sharing Safety: Full Trace Simulation

We further use simulations to reveal the factors that influence sharing safety. The simulations use YARN-CS as the scheduler in the rest of this section. To validate simulation accuracy, the simulator replays the experiments in §5.1 and we compare the obtained job queuing delay to that in §5.1. The largest difference across all the experiments is within 7%. In the simulations we also observe similar sharing anomalies shown in the real experiments, so we believe the variations do not affect our main conclusion.

Queuing delay in a cluster with the original size. The top two figures in Figure 6(a) show the queuing delay for jobs from two representative tenants, prod-a and prod-e, submitted in 20 days. The jobs run in a cluster of the same size as the original production cluster (279 8-GPU nodes). The result is averaged in a 12-hour sliding window over job submission time. In the bottom figure of Figure 6(a) we also show the level of fragmentation of GPU affinity to observe its correlation with queuing delay. At any time, the level of fragmentation is defined as the proportion of 8-GPU nodes that cannot provide 8-GPU affinity for a high-priority job.

Among the three solutions, HiveD achieves the shortest queuing delay in both tenants. Tenant prod-a suffers a longer queuing delay in its private cluster in several time slots (e.g., the first 5 days) when the resource demands exceed its capacity. Both the quota-based scheme and HiveD reduce the queuing delay significantly by running low-priority jobs. However, from Day 11 to Day 12, prod-a experiences a longer queuing delay (200 minutes) in the quota-based cluster than that in the private cluster. In this period, the fact that no queuing delay

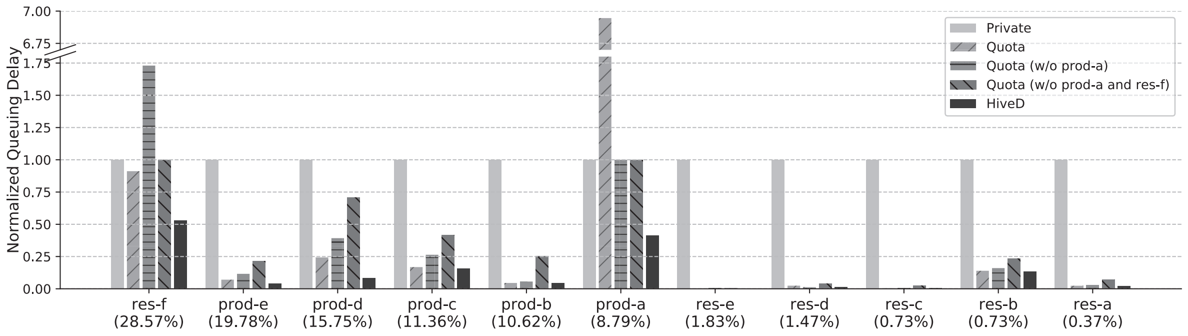


Figure 7: Average queuing delay of each tenant, normalized to that in its private cluster (200 nodes).

observed in its private cluster suggests prod-a has enough GPU quota. But the fragmentation level in the cluster reaches 100%, suggesting the quota-based scheme cannot find even one node to run an 8-GPU job for prod-a. In comparison, prod-a in HiveD has zero queuing delay since it has enough 8-GPU cells available. Overall, the fragmentation level in HiveD is lower than that in the quota-based scheme, because HiveD reserves cells for each tenant, preventing the fragmentation of reserved GPU affinity.

Queuing delay in a higher-load cluster. When a cluster is under-utilized, sharing anomaly is less likely to happen due to sufficient GPU affinity. To further understand the impact of cluster load on sharing safety, we keep the workload unchanged but reduce the cluster size to 200 8-GPU nodes (1,600 GPUs) and rerun the simulation. In this setup, around 90% of the GPUs are used by high-priority jobs. The results are shown in Figure 6(b). In the quota-based scheme, prod-a experiences more severe sharing anomaly when the cluster load is higher. The anomaly lasts from Day 9 to Day 19: the queuing delay can be 8,000 minutes longer than that in the private cluster. The higher cluster load incurs a higher level of GPU affinity fragmentation: the fragmentation level stays at 100% for most of the time, which delays the multi-GPU jobs.

For tenant prod-e, the queuing delays for both Quota and HiveD are always shorter in a shared cluster than in the private cluster. This is because its workload is dominated by a large number of 1-GPU jobs (refer to Table 1), which are immune to the fragmentation of GPU affinity. HiveD can further reduce prod-e's queuing delay by guaranteeing its multi-GPU affinities for its multi-GPU jobs.

We also compare the average queuing delay in the three schemes for each tenant and show the result in Figure 7. The bars marked "Private" and "Quota" show that prod-a's queuing delay in Quota is nearly 7× that in its private cluster. In contrast, the bars marked "HiveD" show that every single tenant has a shorter queuing delay in HiveD than in the private cluster. Compared to Quota, HiveD reduces the queuing delay in 9 out of the 11 tenants (accounting for over 98% quota) due to lower fragmentation level. This reduction is up to 94% (for tenant prod-a), and on average 9% for all the 11 tenants.

In all the previous experiments, the cluster utilization in HiveD is similar to or slightly better than that in quota-based scheme. At some time instances, HiveD improve the utilization over quota-based scheme by up to 20% in the 200-node case and 14% in the 279-node case, as a result of reduced queuing delay. In fact, cluster utilization may depend on the "shape" of jobs (i.e., number of GPUs per job). For example, with a sufficient number of 1-GPU jobs, one can easily saturate the whole cluster. Therefore, our evaluation does not focus on cluster utilization.

Sharing anomalies leading to diminishing benefits of sharing. Figure 7 shows prod-a suffers from severe sharing anomaly (7× queuing delay). It is no longer beneficial for prod-a to contribute its GPUs to the shared cluster. We then run the experiment again to evaluate the effect of decommissioning prod-a (removing its GPUs and workload) from the cluster. The result is shown in the bars marked "Quota (w/o prod-a)" in Figure 7. This time, res-f becomes the victim of sharing anomalies, suffering over 1.7× longer queuing delay. As the largest tenant, res-f previously benefits less (9% shorter queuing delay in Quota than its private cluster) from contributing GPUs to the cluster, compared to the smaller tenants. Because prod-a contains mostly multi-GPU jobs, after decommissioning prod-a, the fragmentation of GPU affinity in the whole cluster becomes worse, leading to longer queuing delay of res-f's multi-GPU jobs and hence the sharing anomaly. This experiment shows the importance of ensuring sharing safety for large tenants. They already benefit less from the shared cluster. They will prefer not contributing their resource to the cluster if experiencing sharing anomaly.

We further decommission res-f from the cluster and rerun the experiment. The result is shown in the bars marked "Quota (w/o prod-a and res-f)" in Figure 7. This time, we do not discover further sharing anomaly. However, the decommissioning of the two tenants greatly reduces the sharing benefits of other tenants. prod-a and res-f contribute 37% of the GPUs in the original cluster. The queuing delay of other tenants in the smaller cluster is clearly longer than that in the larger clusters (before removing prod-a and res-f).

In contrast, with HiveD, not a single tenant suffers from

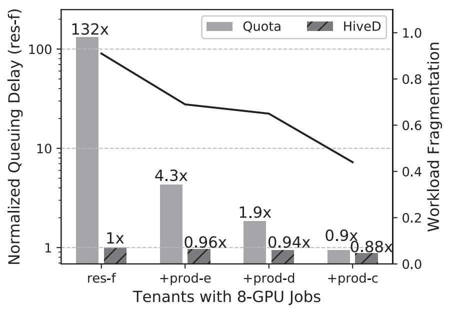


Figure 8: Queuing delay of res-f normalized by private cluster vs. workload fragmentation level (200 nodes).

sharing anomaly in all the settings. And HiveD's queuing delay is consistently shorter than those without HiveD, across all settings. This highlights the necessity of sharing safety.

GPU affinity requirement vs. Sharing safety. We find that the distribution of the GPU affinity requirement for jobs across tenants affects sharing safety. Large number of 1-GPU jobs from other tenants may interfere (or fragment) the GPU affinity of a tenant, leading to sharing anomaly. To show this, we "reshape" the GPU affinity requirement of jobs and observe the queuing delay. In the experiment, we divide the 11 tenants into two groups: jobs in one group are reshaped to the GPU affinity of 1×8 (8-GPU), and those in the other group are changed to 1×1 (1-GPU). The reshaping does not change the total number of GPUs used in each tenant: the number of jobs N is defined by the total number of GPUs divided by the GPU number of a job (8 and 1 in this case). And the job submission time is set by randomly sampling N jobs from the original trace. We further define *workload fragmentation* as the ratio of the total number of jobs to the total number of GPUs. Jobs with higher affinity level have a lower workload fragmentation. The metric will be 1 if all jobs use 1 GPU.

Initially, only res-f is in the 8-GPU group, while the rest tenants go to the 1-GPU group. Figure 8 shows the queuing delay of res-f (normalized by that in its private cluster) and the workload fragmentation, when tenants prod-e, prod-d, and prod-c are moved to the 8-GPU group one by one. When only res-f is in the 8-GPU group, the workload is highly fragmented (0.91). This leads to severe sharing anomaly in Quota-based system: $132\times$ of the queuing delay in the private cluster. When more tenants are moved to the 8-GPU group, the workload fragmentation level goes down. This correspondingly reduces sharing anomaly. res-f experiences shorter queuing delay after the other three tenants are added ($4.3\times$, $1.9\times$, $0.9\times$, respectively, as shown in Figure 8). In contrast, HiveD guarantees sharing safety even under the highest fragmentation and consistently provides shorter queuing delay. Similar trends are observed in other tenants, we hence omit the detailed results here.

Soft affinity requirement. We also study the impact on sharing safety when the GPU affinity requirement of some jobs is soft, i.e., relax the affinity if it cannot be satisfied.

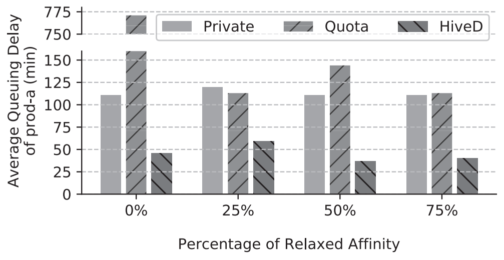


Figure 9: Sharing anomaly still happens when some jobs' GPU affinity requirement is soft.

According to [86], not all training jobs will suffer from performance degradation with relaxed affinity. Hence in the study, we make the most optimistic assumption on the performance degradation: jobs with soft affinity requirement will not sacrifice the training speed. Surprisingly, sharing anomaly could still happen in this case for a quota-based scheme. Figure 9 shows the average queuing delay of tenant prod-a when some multi-GPU jobs in the trace are randomly selected to relax its GPU affinity. We use the 200-node setting in the experiments.

Figure 9 shows that prod-a still has sharing anomaly when 50% of the multi-GPU jobs are allowed to relax their affinity. The average queuing delay in the quota-based scheme is $1.3\times$ of that in its private cluster. Although no obvious anomaly found in the average queuing delay when the job ratio set to 25% and 75%, we still observe sharing anomalies in certain time instances. This is similar to the behaviors in Figures 5(a) and 6. We omit the details due to space limit. On the other hand, HiveD eliminates all the sharing anomalies and always has the shortest queuing delay. Note that Figure 9 shows the best case scenario for relaxed affinity. In reality, jobs with relaxed affinity could perform much worse than the same jobs with the desired affinity [86]. Thus sharing anomaly may happen more likely than it is described in Figure 9.

Although relaxing affinity may reduce the queuing delay for jobs with soft affinity requirement, the behavior may increase the fragmentation of GPU affinity in the cluster. This in turn will increase the queuing delay for jobs with hard affinity requirement. It becomes a complex tradeoff among queuing delay, fragmentation of GPU affinity, training performance, and cluster utilization. HiveD reserves cells to achieve sharing safety and avoids the complex tradeoff altogether.

5.3 Buddy Cell Allocation

In this section, we evaluate the buddy cell allocation algorithm through trace-driven simulations, to understand its effectiveness in reducing preemption and fragmentation of GPU affinity, and its algorithm efficiency.

Reducing preemption with dynamic binding. In the buddy cell allocation algorithm, cells are bound to those in the physical cluster dynamically. This reduces unnecessary preemption of low-priority jobs when there are idle cells. Figure 10 shows the numbers of job preemption when using dynamic

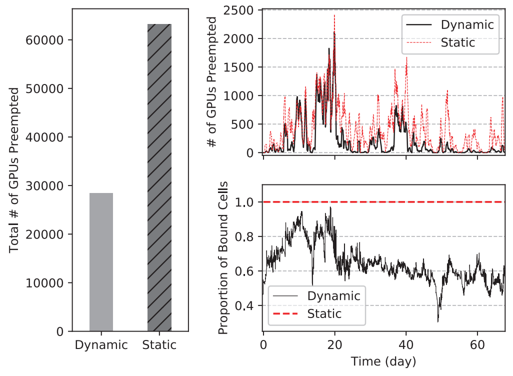


Figure 10: Preemption in dynamic and static bindings.

binding and static binding, respectively. This experiment uses the same setup as the 279-node experiment in §5.2. In total, dynamic binding reduces the number of preempted GPUs by 55%. We also measure the correlation between preemption and the proportion of bound cells in the time dimension (on a 12-hour window). When there are more cells being bound to the physical cluster (e.g., Day 10, Day 20 in dynamic binding), there are also more GPUs being preempted. This is because we have fewer choices of physical cells to bind, hence fewer opportunities to reduce preemption. This observation is also consistent with the fact that static binding, where this proportion is always 100%, incurs many more unnecessary preemptions.

Reducing fragmentation of GPU affinity with multi-level cells. Multi-level cells allow the buddy cell allocation algorithm to pack the cells at the same level across tenants to reduce the fragmentation of GPU affinity. For example, if two tenants both have a level-1 (1-GPU) cell, the algorithm prefers selecting two cells from the same physical node, i.e., buddy cells, to run a 1-GPU job. Instead, if both tenants only reserve level-4 cells (8-GPU, node level), the two tenants have to use a level-4 cell to run its 1-GPU job. Hence the two 1-GPU jobs will be placed on two different nodes, which increases the fragmentation of GPU affinity at node level.

To demonstrate this, instead of only assigning level-4 cells, we assign cells from level-1 to level-4 while keeping the total number of GPUs assigned to each tenant the same as in the above 279-node simulation. Each tenant's assignment matches the distribution of its demands on each level of the cells. Figure 11 shows the fragmentation level of GPU affinity over time when using multi-level and single-level (level-4) cells, respectively. The fragmentation level is always lower with multi-level cells. The gap is more than 10% (up to 20%) for most of the time, which means we can spare roughly 30 more level-4 cells. HiveD therefore recommends that tenants model their job's affinity requirements more precisely, in order for a cluster to perform more efficient cross-VC packing.

Algorithm efficiency. We profile the performance of our

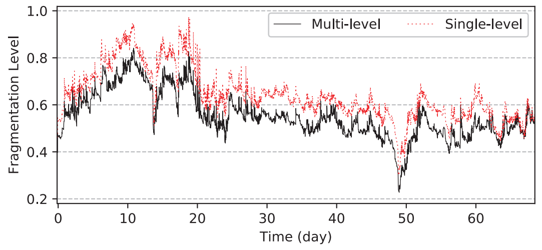


Figure 11: Fragmentation with multi- and single-level cells.

implementation of buddy cell allocation in a setup of a 65,536-GPU cluster with 8 racks, each consisting of 1024 8-GPU nodes. We issued 10,000 cell allocation requests at random levels. The average time to complete a request is 2.18ms. A large part of the cost comes from ordering cells according to low-priority jobs, which accounts for 88% of the time. As the algorithm is clearly not the system bottleneck, we do not perform further optimization (e.g., lock-free operations).

6 Discussion

VC assignment. The VC assignment to a tenant, in terms of both the number of GPUs and their cell structures, has impacts on the effective VC utilization and queuing delay across tenants. The VC assignment is usually a business process, a common practice in large production clusters, e.g., Borg [84]. Factors to consider in VC assignment include overall capacity, tenant demands, composition of tenant workload, workload variation over time, business priority, and budget constraints. Therefore, HiveD leaves the choice of VC assignment to users. In most cases, a tenant can just reserve several node-level cells as a VC and adopt a deep learning scheduler for the VC. If a tenant has more details about workloads, e.g., the GPU number distribution of the jobs, the tenant can reserve different levels of cells to match the job requirement and enjoy less fragmentation and preemption, as discussed in §5.3. VC assignment is a new kind of resource reservation based on cells, and HiveD is a framework to enforce such a reservation.

Job migration. Migrating jobs between GPUs is a powerful mechanism that has been shown effective [86] in improving quality of GPU allocations. De-fragmentation via migration can in theory be used to resolve potential sharing safety violations, but our experience has shown that there are significant challenges in applying migration in production. Fully transparent migration remains challenging in practice, due to implementation issues in different deep learning frameworks (e.g., inconsistent or limited use of certain programming APIs; challenges of multi-language, multi-framework, and multi-version support [21, 30, 62, 65]). Moreover, the choice of which jobs to migrate and where could be rather complex, with different conflicting objectives to balance and a large search space. As shown in §5.1, a greedy migration algorithm [86] can still violate sharing safety. In contrast, HiveD's cell abstraction and buddy cell allocation algorithm enable separation of concerns. HiveD can also leverage migration, especially within each tenant—it will be a search space constrained to within a tenant under the sharing-safety guarantee.

HiveD in the cloud. Major cloud providers are offering GPU VMs in the cloud. Our findings in HiveD are highly relevant even in the cloud setting and can shed light on the types of offering in the cloud. Our buddy cell allocation algorithm can also be used by the cloud providers to manage their reserved [2,6,27] and spot [4,8,78] GPU instances, as our VC cells are essentially reserved instances and our low-priority cells are essentially preemptible spot instances. HiveD's implementation already satisfies requirements of a typical cloud provider, e.g., supports different GPU models, reserves pay-as-you-go instances [70], and handles expansion in capacity. For practical deployment, HiveD can use a hybrid strategy to leverage the cloud as an extension of a multi-tenant GPU cluster when the demand temporarily exceeds the capacity, or can be deployed entirely on a cloud using reserved resources at a lower price, with the options to (i) use spot instances, (ii) buy pay-as-you-go instances when needed, and (iii) purchase and sell reserved capacity in the marketplace [3,23].

Extending HiveD to other affinity-aware resources. Although this paper focuses on reserving affinitized GPUs, HiveD's design applies to other types of affinity-aware resources as well. For example, the cell can be used to define affinitized CPU cores within the same NUMA node, or even multiple types of NUMA-aware resources like affinitized GPUs and CPU cores under the same socket [18].

7 Related Work

Affinity-aware schedulers for deep learning training. Affinity has been well considered something important when scheduling deep learning jobs [22,41,51,52,59,66,72,86] as well as other (big-data) jobs [36,38,89]. HiveD complements these schedulers by applying them in virtualized cluster views, thereby leveraging their efficiency while avoiding sharing anomalies, as identified and shown in our experiments.

Fairness in shared clusters. Identifying the fair share of resources in large clusters has been widely studied. Max-min fairness [55] has been extended in a CPU cluster to address fair allocation of multiple resource types (DRF [37]), job scheduling with locality constraints [38,39,48,89], and correlated and elastic demands (HUG [31]). There are recent proposals to achieve fairness and efficiency for machine learning workloads [29,60,64].

In contrast, HiveD focuses on sharing safety with respect to *given* resource shares (i.e., VC assignment). As we have discussed in §6, determining the resource shares is usually a business process. HiveD assumes a pre-agreed resource partition among multiple tenants, and enforces it with the sharing safety guarantee. This is driven by witnessing that corporate users are annoyed by the uncertain availability of GPU resources that are already assigned to them. In this sense, HiveD is a framework to guarantee a type of resource reservation [91], defined in terms of cells in VCs. HiveD can address fairness by applying the fairness schemes (e.g., Themis [60]) to determine fine-grained fair-share for jobs within a tenant (or across tenants for low-priority jobs), given the coarse-grained VC assignment enforced by HiveD.

Performance isolation. Performance in a shared cluster is sensitive to various sources of interference, including I/O, network, and cache. There are research works on performance isolation that include storage isolation [32,42,43,80], appliance isolation [24,76], network isolation [44,58,67,75,87], and GPU isolation [25,26,50,53,69,88]. In HiveD, we identify a new source of interference: the fragmentation of GPU affinity in a tenant may affect the GPU affinity in other tenants in a shared GPU cluster. To eliminate such interference, HiveD adopts the notion of VC to encapsulate the requirement in multi-level cells and constrains the scheduling behavior within each VC.

Reducing fragmentation. Reducing fragmentation is important to cluster utilization, which has been widely studied in past decades. Tetris [40] is a multi-resource scheduler to pack tasks to avoid resource fragmentation. Feitelson [35] also proposed a buddy-based algorithm to reduce fragmentation for gang-scheduled jobs in supercomputers. There are also works using migration/preemption to reduce fragmentation for gang-scheduled jobs [63,71,86]. HiveD's buddy allocation algorithm with affinity hierarchy can also effectively reduce fragmentation. More importantly, HiveD takes a step further to guarantee sharing safety, i.e., eliminate the external fragmentation across tenants. Ensuring sharing safety requires not only minimizing fragmentation but also explicitly defining cells assigned to each VC, and enforcing this assignment during physical resource allocation.

8 Conclusion

Motivated by observations from production clusters and validated through extensive evaluations, HiveD takes a new approach to meeting the challenge of sharing a multi-tenant GPU cluster for deep learning by (i) defining a simple and practical guarantee, sharing safety, that is easily appreciated by tenants, (ii) proposing an affinity-aware resource abstraction, cell, to model virtual private clusters, (iii) developing an elegant and efficient algorithm, buddy cell allocation, that is proven to guarantee sharing safety and is naturally extended to support low-priority jobs, and (iv) devising a flexible architecture, to incorporate state-of-the-art schedulers for both sharing safety and scheduling efficiency. All these combined, HiveD strikes the right balance between multiple objectives such as sharing safety and cluster utilization.

Acknowledgements

We thank our shepherd Junfeng Yang and the anonymous reviewers for their constructive feedbacks that helped improve the clarity of the paper. We thank Jim Jernigan and Kendall Martin from the Microsoft Grand Central Resources team for providing GPUs for the evaluation of HiveD. Fan Yang thanks the late Pearl, his beloved cat, for her faithful companion during writing this paper. This work was partially supported by the National Natural Science Foundation of China under Grant No. 61972004.

References

[1] Hadoop: Fair scheduler, 2016. https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/FairScheduler.html.

[2] Announcing general availability of azure reserved vm instances. https://bit.ly/2jEFKHR, Nov. 2017.

[3] Amazon EC2 reserved instance marketplace. https://aws.amazon.com/ec2/purchasing-options/reserved-instances/marketplace/, Apr. 2019.

[4] Aws spot instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html, Apr. 2019.

[5] Common voice dataset. http://voice.mozilla.org/, Apr. 2019.

[6] Google cloud: Committed use discounts. https://cloud.google.com/compute/docs/instances/signing-up-committed-use-discounts, Apr. 2019.

[7] Kubernetes default scheduler. https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/, June 2019.

[8] Preemptible virtual machines. https://cloud.google.com/preemptible-vms/, Apr. 2019.

[9] Scheduler extender. https://github.com/kubernetes/community/blob/master/contributors/design-proposals/scheduling/scheduler_extender.md, Jan. 2019.

[10] Statefulsets. https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/, June 2019.

[11] Tiresias code. https://github.com/SymbioticLab/Tiresias/, Feb. 2019.

[12] VCTK dataset. https://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html, Apr. 2019.

[13] Wmt16 dataset. http://www.statmt.org/wmt16/, Apr. 2019.

[14] Amd Radeon Instinct MI50 accelerator. https://www.amd.com/en/products/professional-graphics/instinct-mi50, Apr. 2020.

[15] Azure vm: Nc-series. https://docs.microsoft.com/en-us/azure/virtual-machines/nc-series, 2020.

[16] Azure vm: Nv-series. https://docs.microsoft.com/en-us/azure/virtual-machines/nc-series, 2020.

[17] HiveD scheduler. https://github.com/microsoft/hivedscheduler, 2020.

[18] Kubernetes topology manager. https://kubernetes.io/blog/2020/04/01/kubernetes-1-18-feature-topoloy-manager-beta, 2020.

[19] Nvidia v100 tensor core gpu. https://www.nvidia.com/en-us/data-center/v100/, Apr. 2020.

[20] OpenPAI. https://github.com/Microsoft/pai, 2020.

[21] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[22] Marcelo Amaral, Jordà Polo, David Carrera, Seetharami Seelam, and Malgorzata Steinder. Topology-aware gpu scheduling for learning workloads in cloud environments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 17:1–17:12, New York, NY, USA, 2017. ACM.

[23] Pradeep Ambati, David Irwin, and Prashant Shenoy. No reservations: A first look at amazon's reserved instance marketplace. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association, July 2020.

[24] Sebastian Angel, Hitesh Ballani, Thomas Karagiannis, Greg O'Shea, and Eno Thereska. End-to-end performance isolation through virtual datacenters. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 233–248, Broomfield, CO, 2014.

[25] Rachata Ausavarungnirun, Joshua Landgraf, Vance Miller, Saugata Ghose, Jayneel Gandhi, Christopher J Rossbach, and Onur Mutlu. Mosaic: An application-transparent hardware-software cooperative memory manager for gpus. *arXiv preprint arXiv:1804.11265*, 2018.

[26] Rachata Ausavarungnirun, Vance Miller, Joshua Landgraf, Saugata Ghose, Jayneel Gandhi, Adwait Jog, Christopher J Rossbach, and Onur Mutlu. Mask: Redesigning the gpu memory hierarchy to support multi-application concurrency. In *ACM SIGPLAN Notices*, volume 53, pages 503–518. ACM, 2018.

[27] Jeff Barr. Announcing amazon EC2 reserved instances. `https://aws.amazon.com/blogs/aws/announcing-ec2-reserved-instances/`, Mar. 2009.

[28] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Commun. ACM*, 59(5):50–57, 2016.

[29] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *EUROSYS*, 2020.

[30] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[31] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Stoica. HUG: Multi-resource fairness for correlated and elastic demands. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 407–424, Santa Clara, CA, 2016.

[32] Asaf Cidon, Daniel Rushton, Stephen M Rumble, and Ryan Stutsman. Memshare: a dynamic multi-tenant key-value cache. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 321–334, 2017.

[33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[34] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[35] Dror G Feitelson. Packing schemes for gang scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 89–110. Springer, 1996.

[36] Panagiotis Garefalakis, Konstantinos Karanasos, Peter Pietzuch, Arun Suresh, and Sriram Rao. Medea: Scheduling of long running applications in shared production clusters. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 4:1–4:13, New York, NY, USA, 2018. ACM.

[37] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Nsdi*, volume 11, pages 24–24, 2011.

[38] Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. Choosy: Max-min fair sharing for datacenter jobs with constraints. pages 365–378, 04 2013.

[39] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert NM Watson, and Steven Hand. Firmament: Fast, centralized cluster scheduling at scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 99–115, 2016.

[40] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. *ACM SIGCOMM Computer Communication Review*, 44(4):455–466, 2015.

[41] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, Boston, MA, 2019.

[42] Ajay Gulati, Irfan Ahmad, Carl A Waldspurger, et al. Parda: Proportional allocation of resources for distributed storage access. In *FAST*, volume 9, pages 85–98, 2009.

[43] Ajay Gulati, Arif Merchant, and Peter J Varman. mclock: Handling throughput variability for hypervisor io scheduling. In *OSDI*, volume 10, pages 1–7, 2010.

[44] Chuanxiong Guo, Guohan Lu, Helen J Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International Conference*, page 15. ACM, 2010.

[45] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[47] Walter L Hürsch and Cristina Videira Lopes. Separation of concerns. 1995.

[48] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 261–276. ACM, 2009.

[49] Jeffrey Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, 1981.

[50] Paras Jain, Xiangxi Mo, Ajay Jain, Harikaran Subbaraj, Rehan Sohail Durrani, Alexey Tumanov, Joseph Gonzalez, and Ion Stoica. Dynamic space-time scheduling for gpu inference. *arXiv preprint arXiv:1901.00041*, 2018.

[51] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based parameter propagation for distributed dnn training. In *SysML*, 2019.

[52] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 947–960, Renton, WA, July 2019. USENIX Association.

[53] Angela H Jiang, Daniel L-K Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A Kozuch, Padmanabhan Pillai, David G Andersen, and Gregory R Ganger. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *2018 USENIX Annual Technical Conference (USENIXATC 18)*, pages 29–42, 2018.

[54] Mark S Johnstone and Paul R Wilson. The memory fragmentation problem: Solved? *ACM Sigplan Notices*, 34(3):26–36, 1998.

[55] J. Kay and P. Lauder. A fair share scheduler. *Commun. ACM*, 31(1):44–55, January 1988.

[56] Kenneth C. Knowlton. A fast storage allocator. *Commun. ACM*, 8(10):623–624, October 1965.

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[58] Jeongkeun Lee, Yoshio Turner, Myungjin Lee, Lucian Popa, Sujata Banerjee, Joon-Myung Kang, and Puneet Sharma. Application-driven bandwidth guarantees in datacenters. In *ACM SIGCOMM computer communication review*, volume 44, pages 467–478. ACM, 2014.

[59] Hyeontaek Lim, David G Andersen, and Michael Kaminsky. 3lc: Lightweight and effective traffic compression for distributed machine learning. *arXiv preprint arXiv:1802.07389*, 2018.

[60] 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.

[61] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993.

[62] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577, 2018.

[63] Ioannis A Moschakis and Helen D Karatza. Performance and cost evaluation of gang scheduling in a cloud computing system with job migrations and starvation handling. In *2011 IEEE Symposium on Computers and Communications (ISCC)*, pages 418–423. IEEE, 2011.

[64] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *OSDI*, 2020.

[65] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[66] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 3:1–3:14, New York, NY, USA, 2018. ACM.

[67] Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. Faircloud: sharing the network in cloud computing. *ACM SIGCOMM Computer Communication Review*, 42(4):187–198, 2012.

[68] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[69] Christopher J Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray, and Emmett Witchel. Ptask: operating system abstractions to manage gpus as compute devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 233–248. ACM, 2011.

[70] Margaret Rouse. Pay-as-you-go cloud computing. https://searchstorage.techtarget.com/definition/pay-as-you-go-cloud-computing-\PAYG-cloud-computing, Mar. 2015.

[71] Kittisak Sajjapongse, Xiang Wang, and Michela Becchi. A preemption-based runtime to efficiently schedule multi-process applications on heterogeneous clusters with gpus. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 179–190, 2013.

[72] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. Scaling distributed machine learning with in-network aggregation. *arXiv preprint arXiv:1903.06701*, 2019.

[73] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: flexible, scalable schedulers for large compute clusters. 2013.

[74] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[75] Alan Shieh, Srikanth Kandula, Albert G Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *NSDI*, volume 11, pages 23–23, 2011.

[76] David Shue, Michael J. Freedman, and Anees Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 349–362, Berkeley, CA, USA, 2012.

[77] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[78] Lee Stott. Microsoft azure low-priority virtual machines – take advantage of surplus capacity in azure, Nov. 2017.

[79] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[80] Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis, Antony Rowstron, Tom Talpey, Richard Black, and Timothy Zhu. Ioflow: a software-defined storage architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 182–196. ACM, 2013.

[81] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125, 2016.

[82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[83] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[84] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, page 18. ACM, 2015.

[85] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[86] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 595–610, 2018.

[87] Di Xie, Ning Ding, Y Charlie Hu, and Ramana Kompella. The only constant is change: Incorporating time-varying network reservations in data centers. *ACM SIGCOMM Computer Communication Review*, 42(4):199–210, 2012.

[88] Hangchen Yu and Christopher J Rossbach. Full virtualization for gpus reconsidered. In *Proceedings of the Annual Workshop on Duplicating, Deconstructing, and Debunking*, 2017.

[89] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, pages 265–278. ACM, 2010.

[90] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[91] L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (rsvp). https://tools.ietf.org/html/rfc2205#section-2#page-19, 1997. IETF RFC2205.