Beware of Fragmentation: Scheduling GPU-Sharing Workloads with Fragmentation Gradient Descent

Paper published in USENIX ATC 2023
Qizhen Weng†, Lingyun Yang†, Yinghao Yu*, Wei Wang†,
Xiaochuan Tang*, Guodong Yang*, Liping Zhang*

†HKUST †Alibaba Group
Agenda

• GPU Sharing & Fragmentation in ML Cluster
  • Inefficiency of Existing Approaches
  • The Fragmentation Measure
  • Fragmentation Gradient Descent
  • Implementation and Evaluation
  • Conclusion
ML-as-a-Service (MLaaS) Cloud

All-in-one platform for users using different ML frameworks

Support various workloads: training, inference, evaluation …

ML tasks running in containers scheduled to >1000 GPU servers

GPU underutilization

25-50% GPU utilization in production ML clusters [1-4]

- Most ML tasks cannot fully utilize the capability of modern GPUs

The need for GPU sharing

• GPU sharing lets multiple tasks run on a single GPU  
  • e.g., via DL framework, CUDA API interception, or hardware support (MIG)
• Sharing saves 50% GPUs in Alibaba PAI [1]

Yet, partial GPU allocation results in fragmentation and limits allocation rate

GPU-sharing cluster H with 1.2k nodes, 6.2k GPUs, 8k tasks (Alibaba)

- Fully packed after allocating 92% GPUs, wasting ~500 GPUs
- User experience scheduling failures even with sufficient GPU allocation quotas

![Graph showing CDF of GPU usage]

**Insufficient GPUs**

**Insufficient CPUs (stranded GPU)**
Agenda

- GPU Sharing & Fragmentation in ML Clusters
- Existing Approaches
- The Fragmentation Measure
- Fragmentation Gradient Descent
- Implementation and Evaluation
- Conclusion
Packing improves allocation

• After GPU sharing, "1 GPU" left in idle but not allocatable to Task A
• Mitigate fragmentation with packing
Recap: Multi-Resource Bin-Packing

Task Resource Request

Node Resource Capacity

Best Fit

Dot Product

GPU

Mem

CPU

Task A

Task B

Remaining Resource A

Node A

Node B

Node C

Remaining Resource B

Remaining Resource B

Task C

Best Fit: Verma et al. "Borg" EuroSys '15
Dot Product: Grandl et al. "Tetris" SIGCOMM '14
Does classical multi-resource bin-packing work for GPUs?

How to formulate GPUs into a resource dimension?
Attempt #1

- Pool together a node’s multiple available GPUs into one logical GPU
  - e.g., 2-GPU node with <0.9 GPUs, 0.4 GPUs> => having 1.3 GPUs idle

- Problem:
  - GPU pooling ignores the allocation boundary between GPUs
  - Unable to differentiate the fragmentation on individual GPUs
Attempt #2

- Treat each GPU as an independent resource dimension
  - e.g., 2-GPU node has 3D-resource vector <16 CPUs, 0.9 GPUs, 0.4 GPUs>
- Problem:
  - Choosing the wrong expansion of task resource vectors may block allocation

Task <2 CPUs, 0.5 GPUs>

```plaintext
CPU 16 CPUs
GPU-1 0.9
GPU-2 0.4
```

Task <2 CPUs, 0.5 GPUs, 0 GPUs>

Task <2 CPUs, 0 GPUs, 0.5 GPUs>
Attempt #2

• Treat each GPU as an independent resource dimension
  • e.g., a 2-GPU node with resource vector <16 CPUs, 0.9 GPUs, 0.4 GPUs>

• Problem:
  • Unlike other resources, GPUs are interchangeable!

  A GPU task has a "deformable" resource vector wrt available GPUs on the nodes, invalidating the conventional bin-packing formulation!
Does classical multi-resource bin-packing work for GPUs?

Not for shared GPUs! Need a new approach to address the fragmentation problem of scheduling GPU-sharing workloads.
Agenda

• GPU Sharing & Fragmentation in ML Clusters
• Existing Approaches
• The Fragmentation Measure
• Fragmentation Gradient Descent
• Implementation and Evaluation
• Conclusion
"To Measure is the First Step to Improve"

• How to formally define fragmentation?
  • "You keep using that word. I do not think it means what you think it means."

• How to further reason the sources of fragmentation?
  • Insufficient GPUs, stranded GPUs, or both, how much do they contribute?

• How to efficiently mitigate fragmentation?
  • Simpler and more explainable than using ML techniques
**Fragmentation in absolute term 😞**

**Bad Def.:** "A node is fragmented if and only if it cannot run any task"

1. **Ignorant** of high-demanding workloads (e.g., Task D has no say on fragmentation)

2. **Skyline tasks** (A, B, C) **dominate** others, regardless of their tiny population (0.06% in our traces).
   - Average skyline task demand: <3.2 CPUs, 0.07 GPUs> is far below avg. task demand: <9.4 CPUs, 0.9 GPUs>

3. **Vulnerable** to small workload changes (C -> C’).
   - Unable to differentiate fragmentation sources.
The absolute measure is overly restrictive in fragmentation identification

- Scheduling 8k tasks to 6.2k GPUs

Fragmentation stays at a low level (<5%) throughout the scheduling — Failing to provide early feedbacks to the scheduling quality
A statistical fragmentation measure 😊

The next task to arrive is considered to be randomly sampled from typical workloads.

Fragmentation region
- Q1 & Q2: Insufficient GPU
- Q4: Stranded GPU
- X-axis: Non-GPU tasks

Frag rate: the likelihood that the arriving task falls in frag regions
- Frag rate*: $f_{n}^{\text{GPU}} \cong 1 - \sum_{m \in Q_3} p_m$
  ($p_m \in (0, 1]$: task popularity)
- Frag amount: $F_n^{\text{GPU}} = f_n^{\text{GPU}} R_n^{\text{GPU}}$
- Cluster frag amount: $F_N^{\text{GPU}} = \sum_n F_n^{\text{GPU}}$

* Roughly, as finer-grained calculation should consider fragmentation at per-GPU level. See more in the §3.2 of the paper.
A statistical fragmentation measure 😊

Given the task distribution of a target workload, it measures the expected GPU resources that cannot be allocated:

- Further broken down into different sources of fragmentation: Insufficient GPU (Q2), stranded GPU (Q4), lack both (Q1), non-GPU tasks (X-axis).
A statistical fragmentation measure 😊

- Sensitive to scheduling quality; useful feedback at early stages

- Scheduling: Frag rate $f_{n}^{\text{GPU}} \uparrow$
- Remaining resources $R_{n}^{\text{GPU}} \downarrow$
- Until all remaining resources are unallocatable to any tasks (i.e., Frag rate = 100%).

- Cluster Frag = $\sum_{n}(f_{n}^{\text{GPU}}R_{n}^{\text{GPU}})$ / Total (%): normalized by cluster GPU capacity

Clustering: Xiao et al. “Gandiva" OSDI '18
Packing: Weng et al. “MLaaS" NSDI '22
Agenda

- GPU Sharing & Fragmentation in ML Clusters
- Existing Approaches
- The Fragmentation Measure
- Fragmentation Gradient Descent
- Implementation and Evaluation
- Conclusion
**Fragmentation Gradient Descent (FGD)**

**Heuristic**: schedule tasks towards the **steepest descent** of fragmentation

FGD scores nodes in parallel, thus scaling to large clusters: each decision can be made in **hundred of milliseconds** in cluster with 1200 nodes.

---

**Algorithm 1**: Task scheduling of FGD

<table>
<thead>
<tr>
<th>Input</th>
<th>Cluster $N$, incoming task $m$, target workload $M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Assigned node $n^*$</td>
</tr>
</tbody>
</table>

1. Initialize node score $S \leftarrow \emptyset$, and output $n^* \leftarrow \emptyset$.
2. **parallel for** node $n \in N$ **do**
   1. **if** Insufficient resources $||$ constraints not met **then**
      1. Return $\triangleright$ Filter out unavailable nodes
   2. $n^* \leftarrow \text{AssignTaskToNode}(m, n)$ $\triangleright$ Hypothetically
   3. $\Delta \leftarrow F_{n^*}(M) - F_n(M)$ $\triangleright$ Fragmentation increment
   4. $S \leftarrow S \cup (n, \Delta)$
3. **if** $S \neq \emptyset$ **then**
   1. $n^* \leftarrow \arg \min_{n \in S} \Delta$ $\triangleright$ pick the node with the least $\Delta$.

---

![Diagram](image.png)

**Node A**

- $F_A += 40$

**Node B**

- $F_B += 10$

**Node C**

- $F_C += -20$
A running example of FGD scheduling

2 accessible GPUs

<table>
<thead>
<tr>
<th></th>
<th>Task A</th>
<th>Task B</th>
<th>Task C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5 GPU idle</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1.0 GPU idle</td>
<td>0.7</td>
<td>0</td>
</tr>
</tbody>
</table>

Frag amount: $F_{n\text{GPU}} = f_{n\text{GPU}} \cdot R_{n\text{GPU}}$

1. To GPU A: A will be fragmented to Task A, B, C
   $F_{A\text{GPU}} = 100\% \cdot 0.2 - 33\% \cdot 0.5$

2. To GPU A: A will have no GPU left, thus no fragmentation
   $F_{A\text{GPU}} = 0 - 33\% \cdot 0.5$

3. To GPU A: Impossible
   $F_{A\text{GPU}} = 0 - 0\% \cdot 0.7$

To GPU B: B will be no fragmentation to any Task

To GPU B: B will be fragmented to Task A, B, C

To GPU B: B will have no GPU left, thus no fragmentation

(before alloc.)

(after alloc.)
Agenda

- GPU Sharing & Fragmentation in ML Clusters
- Existing Approaches
- The Fragmentation Measure
- Fragmentation Gradient Descent
- Implementation and Evaluation
- Conclusion
Large-scale trace-driven emulation

- **Implementation**: a pluggable scheduler in Kubernetes
- **High-fidelity event-driven emulator**
  - Cluster-H: 1.2k nodes, 6.2k GPUs
  - Production trace of 8k tasks as input
  - Plugin + Emulator: 10k lines of code

- **FGD** outperforms all heuristics
  1. Has the least amount of GPU fragment
  2. Hosts more tasks before saturation
  3. Packs tasks to 250+ fewer nodes
  4. Reduces unallocated GPUs by 33-49% (utilizes additional 150-290 GPUs)

  - Emulator: [https://github.com/hkust-adsl/kubernetes-scheduler-simulator](https://github.com/hkust-adsl/kubernetes-scheduler-simulator)

  - Clustering: Xiao et al. "Gandiva" OSDI '18
  - Packing: Weng et al. "MLaaS" NSDI '22
Under varying workload distribution

**GPU-Sharing Tasks**

![Graph showing unallocated GPU percentage for varying proportions of GPU-sharing workloads.]

**Multi-GPU Tasks**

![Graph showing unallocated GPU percentage for varying proportions of multi-GPU workloads.]

**Tasks with GPU-type constraints**

![Graph showing unallocated GPU percentage for varying proportions of workloads with GPU type constraints.]

**Non-GPU Tasks**

![Graph showing unallocated GPU percentage for varying proportions of non-GPU workloads.]

27
Agenda

• GPU Sharing & Fragmentation
• Existing Approaches
• The Fragmentation Measure
• Fragmentation Gradient Descent
• Implementation and Evaluation
• Conclusion
Conclusion

Allocating partial GPUs results in severe fragmentation

- A new challenge that cannot be addressed using conventional bin-packing approaches

A new fragmentation metrics

- Measure the expected GPU resources that cannot be allocated given a workload distribution
- Support breakdown analysis to reason about fragmentation

Fragmentation Gradient Descent (FGD)

- Schedules tasks towards the steepest descent of GPU fragmentation
- Packs tasks to fewer nodes, substantially reducing unallocated GPUs
- Easy to implement