2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS) | 979-8-3503-3071-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICPADS60453.2023.00410

Hay: Enhancing GPU Sharing Performance With Two-Level Scheduling for Ray

Lianghong Huang[1], Zejia Lin[1], Wei Liu[2#], Xianwei Zhang[1#]
[1]School of Computer Science & Engineering, Sun Yat-sen University
[2]China Electronic Product Reliability and Environmental Testing Research Institute (CEPREI)
{huanglh59, linzj39}@mail2.sysu.edu.cn, liuwei@ceprei.com, zhangxw79@mail.sysu.edu.cn

***Abstract*—Graphics Processing Units (GPUs) are extensively adopted in many clusters, providing computational services concurrently for applications from a wide spectrum of domains, especially deep learning (DL). To simplify DL training, a unified framework like Ray has been developed to deploy models on scaled clusters. Nevertheless, existing frameworks commonly choose to allocate GPUs to DL training tasks in an exclusive fashion to maximize performance. Inevitably, the dispatched tasks are incapable of occupying the ample GPU resources fully, and even worse the regular jobs are disallowed to co-locate to guarantee exclusiveness. Towards the issue, this paper proposes `Hay`, a resource-aware dynamic scheduler to cooperatively dispatch DL training tasks and regular workloads in GPU clusters. The design tracks the resource of all GPUs in the cluster, and models node capacity by a busyness score computed from corresponding GPUs' utilization. Incoming tasks are processed by a two-phase heuristic policy to select the best node and GPU. Experiment results demonstrate that `Hay` remarkably reduces the interference between GPU-sharing tasks and achieves an average of 1.18x (up to 1.43x) performance improvement compared to the Ray scheduler.**

***Index Terms*—Workload balance, GPU-sharing, Scheduling**

I. Introduction

The last decade has witnessed the excessive growth of computation power in Graphics Processing Units (GPUs) to accelerate a myriad of workloads, including deep learning, data processing, and scientific computing. Multi-task scheduling mechanisms are adopted to coordinate the tasks intra- and inter- GPUs, ensuring minimal resource competition and maximizing hardware utilization. Targeting at allocating resources and dispatching tasks, a plethora of solutions have been proposed, including performance prediction-based schedulers [1]–[3] and GPU sharing techniques [4].

Deep learning is a representative workload in GPU clusters, whose computing requirement fluctuates temporally, especially for distributed training, and long stalls for scattering and reducing data from multiple devices. Ray is a computing framework that handles resource allocation and task placement according to a logical resource requirement parameter set by the user [5], [6]. It assumes training tasks occupy GPUs exclusively, and allocates the entire GPU to each training actor. This simple policy lacks the sharing of GPU resources. To address the issue, techniques to share GPUs between deep learning applications are proposed [4], [7]. These methods are based on the characteristics of the applications themselves, and may not be suitable for sharing GPUs with other tasks.

To jointly schedule training and regular tasks in clusters with load balance and improve GPU utilization, we propose `Hay`, a resource-aware runtime scheduler. We model GPUs as state machines to indicate their availability and quantify the busyness of nodes to account for GPUs' state and node capacity. Our task mapping heuristics prioritize device selection based on the busyness level, enabling a load-balanced compliant placement and facilitating the allocation of supplementary tasks in the future. In the end, with unbiased workloads on GPUs, concurrently executable tasks increase thus improving utilization and performance.

In summary, the contributions of this paper are:

- We highlight the resource wastage scheduling mechanisms of prior arts in scheduling distributed training and regular tasks on GPU clusters, resulting in the under-utilization of GPUs.
- We propose a resource-aware runtime scheduler for striking inter- and intra-node balance, well accounting for hardware utilization and cluster performance, allowing co-executing more tasks for higher throughput.
- Evaluations on real-world applications demonstrate that our design can effectively reduce the run time of both training tasks and regular workloads, outperforming the state-of-the-art.

II. Background & Motivation

A. Ray

Ray is an open-source computing framework for scaling AI and Python workloads. It enables machine learning developers to scale their applications from laptops to production clusters seamlessly without any code change , reduces friction going from development to production, and seamlessly scales applications to ease programming and deployment burdens. Each node has a raylet as a distributed scheduler, responsible for scheduling tasks submitted to this node or spilled to this node by raylets in other nodes. And there is a Global Control Service (GCS) in the cluster, GCS pulls resource availability from each raylet periodically and then aggregates and rebroadcasts them back to each raylet. There is a cluster task scheduler to choose the best node for running this task

#Corresponding author.

2690-5965/23/$31.00 ©2023 IEEE
DOI 10.1109/ICPADS60453.2023.00410

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on October 14,2025 at 03:50:04 UTC from IEEE Xplore. Restrictions apply.

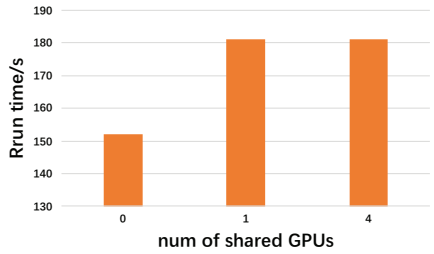


Fig. 1. Comparison of different numbers of shared GPUs in training task. Shared GPU means the GPU running more than one task.

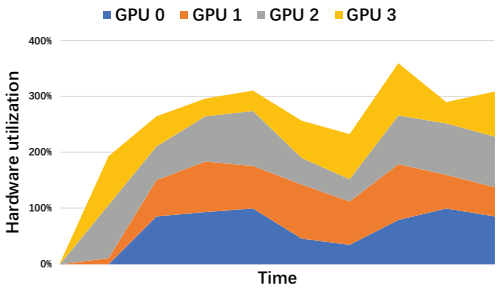


Fig. 2. Accumulated resources usage during training.

and a local task scheduler to try to dispatch and schedule tasks in this node.

B. Motivation

Workload imbalance in DDP. DistributedDataParallel (DDP) training uses several GPUs to train the model with different sampled input data. If part of the GPUs used for training are much slower than the others, gradient synchronization will be the bottleneck in DDP thus leading to performance degradation. In this case, partial GPUs are busy processing tasks, leaving the others to be idle. As shown in Fig. 1, compared to no GPU sharing, the run time of training becomes longer when we have a task sharing a GPU with this training task. However, if we run the same task across all four GPUs, the run time of training tasks will not be lengthened.

Resources wastage of Ray Train. Using Ray for DDP model training allocates the entire GPU for each replica by default. We use Ray Train to train Resnet50 with four A100 GPUs, and the resource utilization is presented in Fig. 2. At certain moments, the utilization of the GPU can reach 100%, but at its lowest, the utilization is only about 30%. Additionally, due to the usage of multiple GPUs, it is challenging for these GPUs to simultaneously achieve 100% utilization.

III. Design

We introduce `Hay`, a novel task scheduler for GPU tasks, primarily addressing the issue of load imbalance among clusters. By considering the load pressure of nodes and each individual GPU, it schedules tasks to keep balance to avoid overload of any GPU or node, so that distributed training

TABLE I
VARIABLE DEFINITION.

Notation	Definition
W_f	Weight of the fragmented GPU resource
W_e	Weight of the entire GPU resource
G_{frag}	Remaining fragmented GPU resource in the node
G_{entire}	Remaining entire GPU resource in the node
G_{total}	Total GPU resource in the node

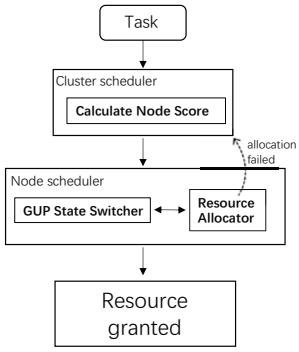


Fig. 3. Architecture overview of `Hay`.

tasks are less likely to have performance degradation due to some GPUs becoming bottleneck. We set different states for the GPUs to grant varying priorities, thus preserving some dedicated GPU resources. Table I introduces the notations we use in this section.

A. Cluster Nodes Workload Balance

To balance the workload among nodes in the cluster, we need to calculate the best node based on the resource information of all nodes. In `Hay`, each node sends its own resource status to the GCS and retrieves information about other nodes from the GCS.

Different nodes in the cluster may have varying counts of GPUs, so it is irrational to calculate the GPU usage of the node to determine its busyness level. It is worth noting that the resource requests can be a combination of fractional GPU requests and entire GPU requests. We should consider the proportion of remaining fragmented GPUs and entire GPUs to score the node. Eq. 1 presents the algorithm to calculate the score of the relative busyness level of the nodes to select the best node for the requests.

$$Score = \frac{W_f * G_{frag} + W_e * G_{entire}}{G_{total}} \tag{1}$$

Since nodes with higher $Score$ tend to have more idle GPU resources, they possibly have the best GPU for the current resource request. We follow a heuristic algorithm to select the optimal node in the cluster task scheduler based on the $Score$, picking up the node with the highest $Score$ as the optimal node. Since every time we schedule the task to a relatively idle node, we can achieve load balance among nodes.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on October 14,2025 at 03:50:04 UTC from IEEE Xplore. Restrictions apply.

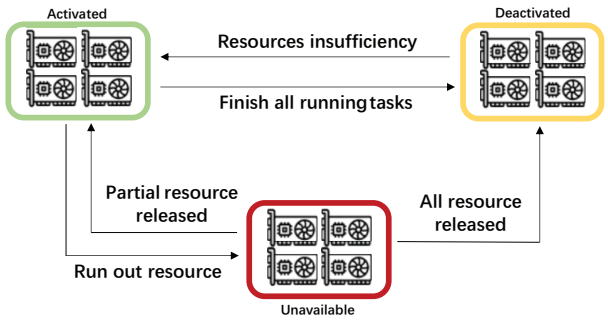


Fig. 4. State machine of GPUs. There are three GPU states in Hay, and these states can transit between each other based on specific conditions.

B. Local GPUs Workload Balance

We aim to balance the workload between GPUs while reserving entire GPU resources as much as possible in this scheduling step. To achieve this goal, we classify GPUs into three states and transform GPU states according to the state machine in Fig. 4. The activated GPUs are ready for any resource request, and the deactivated GPU means that this GPU is idle but we will not directly schedule tasks for this group of GPUs. Deactivated GPUs are ready for tasks that require an entire GPU, or when there is insufficient resource in an activated GPU to be allocated to a task, these GPUs will be activated whenever ready. Unavailable GPUs are the ones that have already been allocated to one or more tasks, and there are no remaining resources. Classifying GPUs in this way enables us to prioritize the fractional GPU resources and reserve the entire GPUs, and by scheduling tasks to the GPU with the most resources we can ensure that the workload is balanced among all activated GPUs. Therefore, there can be more GPU resources in the servers to handle tasks that require the entire GPU, balancing the workload among GPUs.

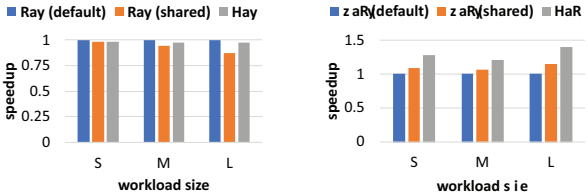
IV. Evaluation

In this section, we conduct experiments to demonstrate the effectiveness of our proposed design Hay. We first illustrate the experimental platform configurations and the applications. We compare our design to Ray both on single-node and multi-node settings.

A. Experimental Setup

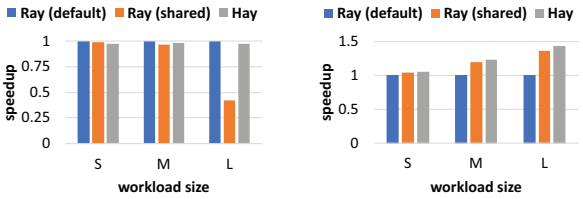
We set up a small cluster with two nodes for testing both intra- and inter- node performance of Hay. Each node in our experimental platform has 8 NVIDIA A100 40GB GPUs and 2 Intel(R) Xeon(R) Gold 6348 CPUs (28 cores).

Applications Resnet50 and mobilenet_v3_small, classical models in deep learning, are used as model training tasks in evaluation. We select applications from NPBench [8] to generate workloads for our experiments. The baseline is using Ray with the default setting, allocating the whole GPU to each training actor. We measure the performance of Ray with GPU sharing and our proposed design Hay. We profile these selected applications ahead of time to decide the resource request of each task in Hay and Ray. The GPU request distribution of the workloads is based on the Alibaba public cluster traces [9].



(a) Training task speedup. (b) Workload speedup.

Fig. 5. Speedup of Resnet50 training tasks and workload in single node experiment. The speedup is normalized to the default setting of Ray.



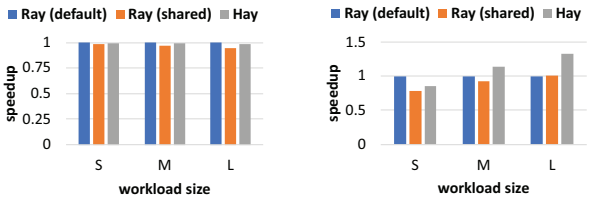
(a) Training task speedup. (b) Workload speedup.

Fig. 6. Speedup of mobilenet_v3_small training tasks and workload in single node experiment. The speedup is normalized to the default setting of Ray.

B. Performance Analysis

1) Intra-node Test: To demonstrate the scheduling ability of multiple GPUs in the same node of Hay under various workload pressures, we generate workloads of size S, M, and L that contain 40, 80, and 160 tasks to share GPU with the training task. The speedups, in terms of normalized execution time, are presented in Fig. 5 and Fig. 6. The result shows that sharing GPUs between training tasks and the normal tasks significantly speeds up the normal ones, with a moderate impact on training tasks. This is because Hay's load balance scheduling preserves more available GPUs for small tasks than Ray. In Fig. 6(a) we observe that training tasks may be severely interfered with when sharing with heavy workloads in Ray, the performance drops to 42.2% compared to the baseline. While Hay not only keeps training performance by a downgrade of less than 3% but also gains a speedup up to 1.4x and 1.43x in normal workload when running with Resnet50 and Mobilenet_v3_small, respectively.

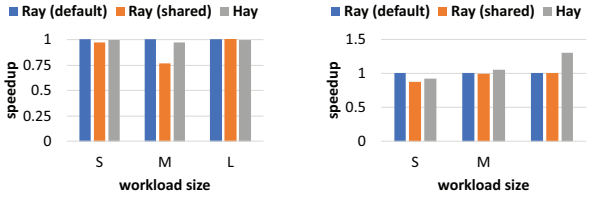
2) Inter-node Test: We also measure the performance of Hay in the inter-node scenario. As shown in Fig. 7 and



(a) Training task speedup. (b) Workload speedup.

Fig. 7. Speedup of Resnet50 training tasks and workload in multi nodes experiment. The speedup is normalized to the default setting of Ray.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on October 14,2025 at 03:50:04 UTC from IEEE Xplore. Restrictions apply.



(a) Training task speedup. (b) Workload speedup.

Fig. 8. Speedup of mobilenet_v3_small training tasks and workload in multi nodes experiment. The speedup is normalized to the default setting of Ray.

Fig. 8, `Hay` also performs remarkably in this scenario. With less than 2% performance decrease in training task, `Hay` achieves up to 1.33x and 1.3x improvement of the workload when running with Resnet50 and Mobilenet_v3_small. The results are similar to the single node test illustrated above, `Hay` has no improvement compared to Ray when dealing with a relatively small workload since the GPU resources are sufficient. However, the results show that `Hay` can schedule large workloads appropriately to minimize the interference with the existing training task and maximize the speedup of workloads.

V. Related Work

Cluster Scheduling. Improving resource utilization [10] and minimizing job completion time [11], [12] are the common objectives of prior designs. Each scheduler is designed with certain purposes, such as solving the scheduling problem for long and short tasks [13], or ensuring the fairness of the scheduler [14], [15]. Furthermore, maximizing job performance [16] and reducing resource fragmentation [17] are also factors that have been well considered. Our work aims to balance the workload among nodes and GPUs, meanwhile preserving as many entire GPUs as possible.

GPU Sharing. Time-sharing GPU is the default way to share a GPU, but it usually comes with a non-negligible overhead of context switch, so some research works are proposed to reduce the overhead [7]. And Nvidia officially provides MPS and MIG for spatial sharing. Some works share the GPU accounting for the characteristics of tasks [18], [19]. Spatial sharing for deep learning applications is also being extensively researched [20], [21]. Our proposed design is based on time-sharing the GPU, we advocate scheduling tasks based on the resource usage to minimize the interference to model training tasks.

VI. Conclusions

In this paper, we identify the problem of under-utilization of GPU during distributed model training in Ray and discover the workload imbalance during distributed training leads to wastage of GPU resources. To address this problem, we propose `Hay`, a scheduler to balance workload among GPUs and reduce competition of shared GPU tasks on training tasks. Our inter- and intra-node experiment results show that `Hay` significantly reduces the interference among co-located tasks, and outperforms Ray on shared GPU workload by 1.18x (up to 1.43x) with trivial impact on training tasks.

Acknowledgment

We thank the anonymous reviewers for their constructive comments and suggestions. This research was supported by the National Natural Science Foundation of China-#62102465, the Funding by Science and Technology Projects in Guangzhou-#202201011241, and Open Project of China Electronic Product Reliability and Environmental Testing Research Institute (CEPREI)-#HK202201334.

References

[1] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *INFOCOM*, 2018.
[2] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking beyond GPUs for DNN scheduling on multi-tenant clusters," in *OSDI*, 2022.
[3] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "Slaq: quality-driven scheduling for distributed machine learning," in *SoCC*, 2017.
[4] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "AntMan: Dynamic scaling on GPU clusters for deep learning," in *OSDI*, 2020.
[5] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan *et al.*, "Ray: A distributed framework for emerging AI applications," in *OSDI*, 2018.
[6] S. Wang, E. Liang, E. Oakes, B. Hindman, F. S. Luan, A. Cheng, and I. Stoica, "Ownership: A distributed futures system for fine-grained tasks," in *NSDI*, 2021.
[7] Z. Bai, Z. Zhang, Y. Zhu, and X. Jin, "PipeSwitch: Fast pipelined context switching for deep learning applications," in *OSDI*, 2020.
[8] A. N. Ziogas, T. Ben-Nun, T. Schneider, and T. Hoefler, "NPBench: A benchmarking suite for high-performance NumPy," in *ICS*, 2021.
[9] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *NSDI*, 2022.
[10] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin, "Multi-resource interleaving for deep learning training," in *SIGCOMM*, 2022.
[11] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A GPU cluster manager for distributed deep learning," in *NSDI*, 2019.
[12] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *EuroSys*, 2018.
[13] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *USENIX ATC*, 2015.
[14] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and efficient GPU cluster scheduling," in *NSDI*, 2020.
[15] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, and S. Viswanatha, "Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning," in *EuroSys*, 2020.
[16] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *OSDI*, 2020.
[17] Q. Weng, L. Yang, Y. Yu, W. Wang, X. Tang, G. Yang, and L. Zhang, "Beware of fragmentation: Scheduling GPU-sharing workloads with fragmentation gradient descent," in *USENIX ATC*, 2023.
[18] Y. Weng, T. Ge, X. Zhang, X. Zhang, and Y. Lu, "RAISE: Efficient GPU resource management via hybrid scheduling," in *CCGrid*, 2022.
[19] X. S. Tan, P. Golikov, N. Vijaykumar, and G. Pekhimenko, "GPUPool: A holistic approach to fine-grained gpu sharing in the cloud," in *PACT*, 2022.
[20] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Serving heterogeneous machine learning models on multi-GPU servers with spatio-temporal sharing," in *USENIX ATC*, 2022.
[21] M. Chow, A. Jahanshahi, and D. Wong, "KRISP: Enabling kernel-wise right-sizing for spatial partitioned gpu inference servers," in *HPCA*, 2023.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on October 14,2025 at 03:50:04 UTC from IEEE Xplore. Restrictions apply.