What are the different software methods that can be employed to effectively share GPU-constrained resources within Kubernetes environments?

Alexis Allemann
alexis.allemann@hes-so.ch
HES-SO Switzerland
Lausanne, VD, CH

Olivier D'Ancona
olivier.dancona@hes-so.ch
HES-SO Switzerland
Lausanne, VD, CH

Magali Egger
magali.egger@hes-so.ch
HES-SO Switzerland
Lausanne, VD, CH

ABSTRACT

Managing GPU resources efficiently within Kubernetes environments presents significant challenges in today's computing landscape, especially with the increasing demand for machine learning tasks. This paper investigates various software solutions dedicated to overcome the difficulties associated with GPU-constrained resource sharing. We explore scheduling frameworks and algorithms tailored for GPU resource management, distinguishing between rule-based and machine learning-based approaches. To accomplish this, we review the latest developments in scheduling algorithms and frameworks, examining their objectives, methodologies, and results. Our methodology evaluates existing literature, by categorizing schedulers based on multiple criteria such as resource utilization efficiency, fairness in resource distribution and elasticity. The results present various schedulers, mentioning their strategies, goals and specifications. Consequently, this paper offers various solutions, letting the user select the most suitable one to suit their requirements.

KEYWORDS

Kubernetes, GPU, Schedulers, Constrained resources

ACM Reference Format:
Alexis Allemann, Olivier D'Ancona, and Magali Egger. 2024. What are the different software methods that can be employed to effectively share GPU-constrained resources within Kubernetes environments?. In *Proceedings of New Technologies in Data Engineering 2024 (NTDE24).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In today's computing landscape, workloads have expanded significantly, especially with the rise of machine learning tasks, which often demand substantial computing power. As a result, there is a pressing need for effective solutions to access and share resources efficiently.

This paper focuses on the challenges of managing graphics processing units (GPUs) constrained resources within Kubernetes environments. 'GPU-constrained resources' refer to scenarios where the availability of GPUs is limited relative to the demands of applications. It includes situations where the parallel processing capabilities of GPUs could be more utilized or managed, leading to inefficient allocation and wastage of resources.

Schedulers are central in this management process, serving as software components responsible for resource allocation. The challenge is ensuring these schedulers operate efficiently and address limitations such as GPU underutilization, resource wastage, memory constraints, and communication costs [10]. Therefore, the primary target of this paper is to investigate various software solutions customized to overcome the challenges associated with GPU-constrained resource sharing.

The paper is organized as follows. In Section 2, we explore the importance of GPU computing, the challenges GPUs face, an introduction to Kubernetes, and how scheduling plays a role in Kubernetes setups. The methodology is outlined in Section 3. In Section 4, we distinguish between scheduling frameworks and algorithms, explaining how they allocate resources. We explore various frameworks and scheduling algorithms. Section 5 covers the metrics used to evaluate scheduler performance. In Section 6, we review the latest developments in scheduling algorithms and frameworks. Section 7 summarises the schedulers analysed and provides a table to help you select an appropriate scheduler according to predefined criteria. Section 8 speculates on potential enhancements for GPU resource sharing and describe the problem of the lack of scientific publications in the field. Finally, Section 9 summarizes the key findings.

2 BACKGROUND

2.1 Importance of GPU Computing

In modern computing, GPUs have assumed a considerable role across various domains due to their ability to accelerate various tasks. This ability comes from their capacity to parallelize matrix multiplication. Specifically, GPUs are indispensable in training neural networks, including sophisticated architectures like deep learning models [19] and transformers [38] to speed up training processes significantly. Furthermore, GPUs play a major role in 3D graphics rendering, ensuring smooth and realistic visual experiences in applications such as video games, virtual reality, and computer-aided design software. They also contribute significantly to video editing tasks, enabling efficient processing and manipulation of high-resolution video files through GPU acceleration, facilitating real-time previews and faster rendering. Additionally, GPUs are essential for cryptocurrency mining [22], particularly in proof-of-work blockchain-based currencies, where they excel in performing the repetitive mathematical calculations necessary for mining operations. Moreover, in scientific research, GPUs are indispensable for conducting complex simulations and computations, with supercomputers and high-performance computing clusters leveraging GPU accelerators to tackle computationally intensive tasks across disciplines such as physics [29], chemistry [36], and biology [33].

2.2 GPU Computational Bottlenecks

The computational bottlenecks of GPUs cover various factors [23] [24] [39], including memory limitations, bandwidth constraints, computing capabilities, and frequency considerations. Memory constraints arise from limited onboard video memory. In case of machine learning, it restricts the batch or models size that can be processed. Bandwidth limitations occur between the GPU cores and the GPU memory, affecting data transfer rates and, thus, overall processing speed. When data transfer rates are suboptimal, the processing speed slows as the GPU cores wait longer for data transfer. Furthermore, variations in computing capabilities and frequencies across different GPU architectures can impact parallel processing efficiency. Notably, inefficient scheduling exacerbates these bottlenecks, resulting in GPU underutilization, resource wastage, memory constraints, and increased communication costs. Therefore, effective scheduling mechanisms are imperative for optimizing GPU resource allocation.

2.3 Kubernetes

Kubernetes [28] is a container orchestration platform designed to automate deployment, scaling, and management of containerized applications. It handles the underlying infrastructure and provides a consistent set of APIs for managing applications across diverse environments. At its core, Kubernetes manages the lifecycle of containers, handling tasks such as scheduling, scaling, and updating. It schedules containers onto a cluster of machines called nodes, ensuring the desired state of the application is maintained.

Figure 1 illustrates the architecture of a Kubernetes cluster, showing the primary components involved in managing containerized applications. The Kubernetes cluster is divided into two main sections: the Control Plane and the Nodes. The Control Plane is responsible for managing the overall state of the cluster and includes the following components: the API Server, which is the central management entity exposing the Kubernetes API and serving as the main entry point for all administrative tasks; Etcd, a distributed key-value store used for storing all cluster data, ensuring consistency and high availability; the Controller Manager, which runs controller processes regulating the state of the cluster, such as replication controllers, node controllers, and endpoint controllers; the Scheduler, which assigns workloads to nodes based on resource availability and other policies; and the Cloud Controller Manager, which integrates the cluster with the cloud provider's API, allowing the management of cloud-specific resources. The Nodes are the worker machines where the containerized applications run, each containing a kubelet, an agent that ensures containers are running as expected and communicates with the control plane, and kube-proxy, which maintains network rules and allows communication to and from the pods.

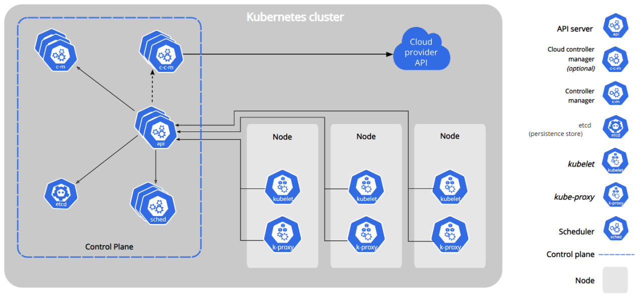

Figure 1: Kubernetes Architecture [1]

One of Kubernetes's key features is its ability to provide service discovery and load balancing, facilitating communication between containers and routing traffic to the appropriate instances of an application. It also facilitates storage orchestration, oversees volume management, and ensures persistent application storage. Kubernetes offers automatic scaling capabilities based on resource utilization. It monitors the health of containers and nodes, automatically restarting containers on healthy nodes or rescheduling them onto available nodes in case of failures.

Resource allocation is essential in container orchestration platforms. It involves managing requests and limits for each container regarding computing capacity and memory, optimizing resource utilization and avoiding contention [6].

2.4 Scheduling in Kubernetes

The Kubernetes scheduler plays an important role in determining which node within the cluster should run each newly created pod, the smallest deployable unit in Kubernetes. This decision is based on factors such as resource requirements, node capacity, and any constraints specified by the user.

The main interface for managing cluster resources is the Kubernetes API server. Users or controllers provide pod specifications to the API server, which then presents them to the scheduler for node assignment. The scheduler selects a node for a pod and updates the API server with the scheduling decision [2].

By default, Kubernetes employs the kube-scheduler algorithm, which evaluates nodes based on factors like available CPU and memory resources and pod-specific constraints and requirements. However, this default scheduler does not differentiate between the characteristics of GPU devices. This is a strong limitation because the default Kubernetes scheduler cannot handle GPU sharing, leading to underexploitation of resources[3]. To address these issues and enhance resource utilization and overall system performance, alternative scheduling mechanisms tailored for GPU workloads are therefore necessary [31].

3 METHODOLOGY

To address our research question, we began with a comprehensive review of existing literature. We opted for a systematic evaluation of the framework strategies and algorithmic solutions. We aimed to encompass a wide range of approaches, from hardware-level modifications to intricate scheduling algorithms. Then, to effectively categorize and analyze the collected literature, we established criteria that reflect the core aspects of GPU resource management. These criteria include resource utilization efficiency, fairness in resource distribution, scalability, and adaptability to dynamic workloads. With our criteria firmly in place, we carefully filter and select research papers that offer significant insights or novel contributions to GPU scheduling within Kubernetes. This selection process was guided by the relevance of each paper's methodology and findings to the challenges identified in the introductory sections of our study. The core of our methodology involves a comparative analysis of the selected papers. We systematically assess how each scheduler addresses the challenges of GPU resource management, comparing their approaches based on our predefined criteria. This analysis is facilitated by a detailed table categorizing each paper's objectives, strategies, algorithms, experimental setups, and key findings.

4 SCHEDULERS DESIGN

4.1 Scheduling frameworks VS scheduling algorithms

It is important to distinguish between scheduling frameworks and scheduling algorithms.

Scheduling frameworks refer to global systems that define how scheduling decisions are made and implemented within a computing environment. These frameworks provide the structure and guidelines for scheduling tasks or resources, often incorporating various scheduling algorithms to achieve specific goals or objectives.

On the other hand, scheduling algorithms are the specific techniques used within a scheduling framework to make individual scheduling decisions. These algorithms determine how resources are allocated, prioritized, and managed within the system. Scheduling algorithms can vary significantly in complexity, efficiency, and suitability for different workload scenarios.

These components work together to optimize resource allocation, improve system performance, and meet the diverse needs of modern computing environments.

4.2 Scheduler strategies

Several strategies can effectively handle GPU-limited resources in Kubernetes by addressing various aspects of resource allocation and workload management. These strategies focus on specific objectives such as exclusive allocation, GPU sharing, group scheduling, elastic training, consolidated placement and topology agnostic placement, as reported in the literature [10]. Figure 2 visually illustrates the distinctions between these strategies.

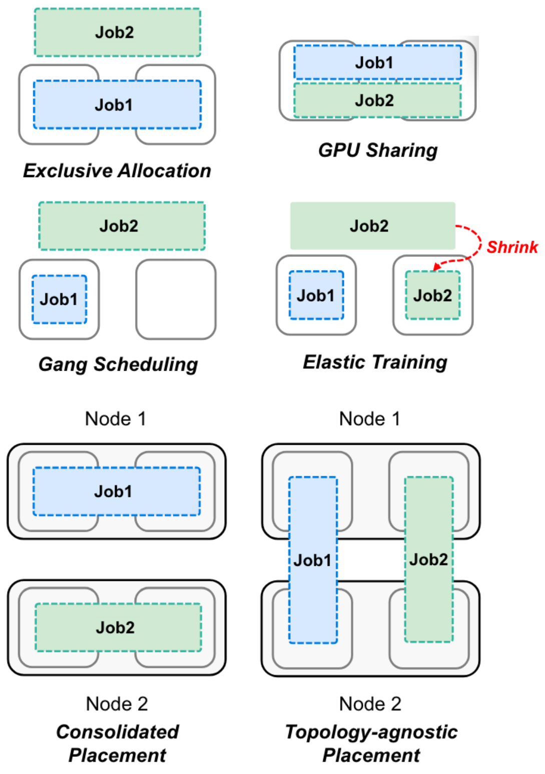

Figure 2: Illustration of the different GPU scheduling frameworks strategies [10]

Exclusive Allocation represents the ultimate in isolation, dedicating entire GPUs to specific workloads. This approach guarantees uninterrupted access to GPU resources for individual tasks, ensuring maximum performance without conflict with other workloads. However, it can lead to under-utilization of GPU resources, particularly when tasks do not fully utilize the allocated resources throughout their execution.

GPU Sharing, on the other hand, allows GPU resources to be used simultaneously by multiple workloads. By enabling multiple tasks to share GPU resources, GPU sharing optimizes resource utilization and can potentially increase overall system throughput. However, this approach can introduce conflicts between competing workloads, resulting in performance degradation or unpredictable execution times.

Gang Scheduling forms groups of GPUs and schedules related tasks on these predefined GPU groups. This method aims to minimize idle time and improve resource utilization by ensuring fairness between competing workloads. However, it does not dynamically adjust resources based on workload demand changes, unlike Elastic Training.

Elastic Training dynamically adjusts GPU resource allocation according to workload demands. This adaptive strategy ensures efficient scaling, allowing resources to be dynamically allocated or deallocated in response to workload fluctuations, optimizing resource utilization. This includes the ability to "shrink" a job by reducing its allocated resources when it needs less, which Gang Scheduling does not support.

Consolidated Placement aims to optimize resource utilization by grouping related workloads on the same physical or virtual GPU. By grouping tasks with similar resource requirements or dependencies, consolidated placement minimizes resource fragmentation and improves overall efficiency. This approach is particularly beneficial in environments where workloads are diverse and fluctuate.

Finally, Topology-Agnostic Placement separates workload placement decisions from the underlying hardware topology. This strategy offers great flexibility in heterogeneous environments, allowing workloads to be placed on available resources regardless of the physical or logical topology of the underlying hardware, thus optimizing resource utilization and improving the flexibility of workload placement.

4.3 Scheduling algorithms types

Scheduling algorithms can be classified into two main categories.

Rule-based schedulers make scheduling decisions based on predefined rules and policies. These rules are typically based on workload priorities, resource constraints, affinity or anti-affinity requirements, and fairness criteria. For instance, a rule-based scheduler may prioritize high-priority workloads over lower-priority ones or ensure that specific workloads are co-located or isolated from each other by enforcing constraints. While rule-based schedulers offer simplicity and predictability, they may lack adaptability to dynamic workload conditions.

Machine learning-based schedulers leverage algorithms and models to make scheduling decisions. These schedulers analyze historical workload data, resource utilization patterns, and other relevant metrics to predict future resource demands. ML-based schedulers can dynamically adapt to changing workload conditions and optimize resource allocation by utilizing techniques such as regression, clustering, or neural networks. For example, they may predict the likelihood of resource contention and adjust scheduling decisions accordingly to improve overall system performance.

4.4 Job execution model

In a Kubernetes environment, deploying a job involves several steps categorized into three main phases. The first phase, the "Pending phase", begins when a job is submitted and lasts until the pod is assigned to a node. It includes the initial wait time before scheduling starts and the time to schedule the pod to a suitable node. Once scheduled, the job enters the "Running phase", which can either lead to successful completion or end in failure. This phase includes the time to download the necessary Docker image and create the container, followed by the execution time of the task. After completing the task, the job enters the final "Deletion phase", where resources are cleaned up. These steps are systematically detailed in KubCG's study [9]. Figure 3 illustrates a job's lifecycle.

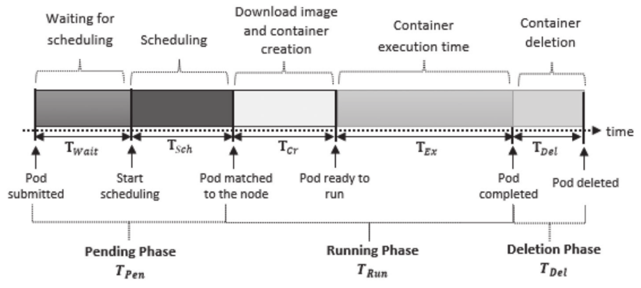

Figure 3: Illustration of a job's lifecycle [9]

5 EVALUATION METRICS

Several metrics are commonly used to assess schedulers effectiveness in resource allocation and optimization [5]. These metrics, gathered throughout our literature review, provide insights into various aspects of scheduler performance.

Resource utilization measures the extent to which available resources, such as bandwidth, memory, and GPU, are utilized by the scheduled workloads. High resource utilization indicates efficient allocation, while low utilization may suggest underutilization or resource wastage.

Fairness metric assesses the equitable distribution of resources among different users or workloads. The objective is to prevent any single workload from monopolizing available resources to the detriment of others. For example, finish-time fairness measures the ratio of a workload's completion time in a shared environment to its completion time if run alone, providing a measure of resource allocation equity [21].

Overall job completion time measures the total duration required to execute a task or workload across a system. This metric includes the time a job spends waiting (pending time), actively processing (running time), and concluding operations (destruction time) [9]. This metric is particularly significant when jobs are distributed across multiple servers. Even if a task runs slower on an individual server, parallel processing on several servers can reduce the aggregate completion time, enhancing performance across the system [45].

Elasticity measures the ability of the scheduler to handle increasing workload demands and scale resources accordingly. An elastic scheduler can manage resource allocation effectively even as workload size and complexity grow. It optimizes resource utilization by scaling resources up or down in response to workload changes, ensuring efficient operation even during peak usage periods.

Evaluating GPU schedulers is a multi-metrics optimization problem that involves trade-offs. While ideally, a scheduler should excel in all performance dimensions, optimizing one metric can sometimes negatively impact another. For instance, optimizing the resource utilization can lead to reduced throughput. Therefore, the objective should not be to maximize isolated metrics but to find the most relevant compromise that best suits your needs.

6 STATE OF THE ART

This section presents important scheduler in the chronological order to depict the scheduling lanscape. Starting around 2018, the first schedulers for GPUs in Kubernetes emerged. The initial part discusses five outdated schedulers, which, despite being obsolete now, mark significant milestones in scheduler development.

In January 2018, a Online Job Scheduling Algorithm for Distributed Machine Learning Clusters, named OASiS, [4] was introduced. OASiS, an online algorithm for scheduling incoming jobs and determining the optimal number of concurrent workers, aimed to efficiently allocate resources in distributed machine learning environments, optimize server utilization, and promptly complete model training tasks. It addressed the challenge of dynamically adjusting the number of concurrent workers and parameter servers for each job based on its progress, maximizing overall utility while considering completion times.

In April 2018, Optimus [26], a job scheduler specifically designed for deep learning clusters, emerged as a significant advancement. Building upon the foundations laid by prior work in distributed machine learning environments, Optimus focused on deep learning training jobs' unique resource requirements and characteristics. It introduced online fitting techniques to predict model convergence during training and set up performance models to accurately estimate training speed based on allocated resources. Optimus aimed to minimize job completion time and maximize cluster efficiency by dynamically adjusting resource allocation and task placement.

In December 2018, Gaia Scheduler presented a topology-based GPU scheduling framework for Kubernetes [35]. It addressed the issue of uneven GPU resource utilization by dynamically adjusting resource allocation based on GPU cluster topology. By restoring the GPU cluster topology in a resource access cost tree and dynamically adjusting resource allocation, Gaia Scheduler aimed to improve load balance and resource utilization, ultimately increasing GPU cluster resource utilization by about 10%.

In September 2019, Kube-Knots introduced Knots, a GPU-aware resource orchestration layer integrated with Kubernetes, forming Kube-Knots [37]. It focused on dynamically provisioning GPU resources based on application needs to improve resource utilization. Kube-Knots introduced scheduling techniques like correlation based prediction and peak prediction to improve GPU utilization and job completion times significantly. Achieving up to 80% improvement in GPU utilization and 36% improvement in job completion times compared to existing schedulers, Kube-Knots led to energy savings and improved Quality of Service for latency-critical queries.

In September 2019, the DL^2 scheduler was introduced as a deep-learning driven scheduler for DL clusters, targeting global training job acceleration [27]. It utilized a combination of supervised learning and reinforcement learning to resize resources allocated to deep-learning jobs dynamically. DL^2 outperformed existing schedulers like Optimus regarding average job completion time, achieving improvements of 44.1% and 17.5%, respectively. By adopting deep-learning techniques, DL^2 provided a generic and efficient scheduler for deep-learning clusters, demonstrating significant advancements in resource scheduling for machine-learning workloads.

In this second part of the section, we explore more in detail the most cutting-edge and innovative scheduler available to this date.

In April 2020, Gandiva_{fair}[7] was introduced as a novel cluster scheduling framework. This scheduler operates as a gang-aware fair scheduler, where "gang-aware" refers to the scheduler's ability to consider and manage groups or "gangs" of related jobs or tasks within the server. Gandiva_{fair} integrates both load-balanced allocations at the cluster level and fairness mechanisms at the server level. Its primary objectives include ensuring fair allocation of cluster-wide GPU time among users, maintaining progress for all jobs, and transparently managing GPU heterogeneity. Testing involved workload simulations and experiments to evaluate performance under varying conditions. Results demonstrated improved fairness, progress, and efficiency in GPU resource utilization compared to existing methods. Notably, Gandiva_{fair} employs a ticketing mechanism to distribute GPU resources fairly among users and leverages gang-aware scheduling to ensure efficient utilization and progress for all jobs.

In June 2020, KubeShare[43] was introduced as a new framework for managing GPU resources within Kubernetes clusters. It introduces the concept of "sharePods" and "vGPUs," allowing for fine-grained allocation and efficient utilization of GPU resources. SharePods represent pods with the capability to attach shared custom GPU devices to their containers, while vGPUs denote the shared GPUs managed by KubeShare. This framework enhances GPU resource management by treating GPUs as first-class resources, enabling users to specify scheduling requirements and constraints. By decoupling scheduling and GPU sharing logic, KubeShare offers flexibility and compatibility with existing Kubernetes setups. Furthermore, its token-based time-sharing mechanism ensures GPU usage isolation and elastic allocation, optimizing resource utilization for diverse workloads. According to the paper, evaluations on real deep learning workloads have shown that KubeShare can significantly increase GPU utilization and overall system throughput by around 2x with less than 10% performance overhead during container initialization and execution.

In November 2020, a highly efficient GPU scheduling framework [40] was developed by wang et al. to optimize deep learning training workloads. This framework introduces several key innovations. Firstly, it features an adaptive scheduler that dynamically allocates and reallocates GPUs based on real-time training job progress. This scheduler can reduce and increase GPU allocations as needed, thereby minimizing job completion times. Secondly, an elastic GPU allocation mechanism complements the scheduler by enabling seamless GPU reallocation without interrupting ongoing tasks. This mechanism, known as the "SideCar" process, ensures lightweight and non-intrusive adjustments to GPU allocations. Combined, these components form a comprehensive GPU scheduling framework integrated into Kubernetes through plugins. Compared to existing solutions like Gandiva [7], which require intrusive modifications to training frameworks, this solution prioritizes reducing completion times without altering underlying code. Evaluation of the framework, conducted on two 16-GPU clusters using TensorFlow-based training jobs, demonstrates significant improvements compared to the default Kubernetes scheduler. The overall execution time and average job completion time are reduced by up to 45% and 63%, respectively.

Still in November 2020, a new resource reservation called HiveD [45] was introduced to enable sharing of a GPU cluster. The framework tackles the "sharing anomaly" problem in multi-tenant clusters. For instance, when a user reserves some GPU resources as part of their quota but cannot use them because of bad fragmentation caused by a scheduling policy. HiveD's approach to eliminating the "sharing anomaly" problem is to separate sharing anomaly from other resource allocation objectives. They designed a solution that guarantees sharing safety by design, distributing virtual cells in private clusters that assert the quota, especially the resources affinity. Thanks to that structure, they can incorporate any state-of-the-art scheduler on top of their architecture. To assess the framework's effectiveness, they evaluate HiveD through diverse experiments on a 96-GPU cluster in a public cloud and trace-driven simulations on a production workload. To sum up, HiveD can incorporate state-of-the-art deep learning schedulers and complement them with sharing safety.

In September 2021, Vapor [46], a GPU-sharing scheduler for distributed deep learning, was introduced. It aims to improve cluster utilization by minimizing interference through overlapping computation and communication tasks. Vapor was developed on the Kubernetes platform and supports the mainstream TensorFlow deep learning framework. Vapor introduces two main improvements: preemptive GPU sharing and adaptive batch redistribution. Preemptive GPU sharing minimizes interference among jobs by parallelizing computation and communication tasks. It isolates computing resources by preempting processes and uses greedy task placement to manage the pipeline. Adaptive batch redistribution deals with performance lags by transferring workloads from heavily loaded to lightly loaded processes. It adjusts batch sizes based on an AIMD [17]model to quickly balance clusters in dynamic environments. Compared to other solutions [44] [32] [30], Vapor focuses on efficient job placement to reduce interference. In tests (conducted on a Kubernetes cluster with 16 Tesla V100 GPUs handling popular DDL jobs), it outperformed Gandiva [7], a fairness-focused solution, that showed limited improvement through packing compared to Vapor, which completes tasks in less time. When combined and implemented on Kubernetes with TensorFlow support, Vapor significantly improves cluster utilization compared to previously stated schedulers.

In October 2021, a new multitasking scheduler called K-Scheduler [18] was introduced. This scheduler tries to enhance the performance of concurrent execution on GPUs by considering the resource utilization patterns of applications running simultaneously. The primary goal of K-Scheduler is to improve the performance of concurrent execution. To achieve this goal, K-Scheduler first analyzes the resource utilization and multitasking behavior of individual applications, considering their classification [48] and execution characteristics. It employs a combination of dynamic and static rules to derive scheduling decisions based on observed patterns. The performance of the K-Scheduler is evaluated using metrics such as Weighted Speedup, which measures the total execution time normalized against sequential execution time and Average Normalized Turnaround Time. Additionally, fairness is assessed using a metric that calculates the difference between the minimum and maximum speedups achieved by tasks. Higher fairness values, closer to 1, indicate more balanced task performance. Other studies [8] [41] [25] focus on partitioning streaming multiprocessor resources, struggling with performance among heavy client requests. In contrast, the K-Scheduler classifies applications by execution traits and considers multitasking dynamics. It ensures individual performance and boosts execution by sidestepping resource contention through multitasking-derived rules.

In December 2021, KubFBS [20], a fine-grained and balance-aware scheduling system for containerized deep learning tasks on Kubernetes clusters, was proposed. KubFBS addressed the limitations of previously existing scheduling strategies by implementing a fine-grained scheduling model that considers multiple GPU-related factors and a balance-aware scheduling objective that breaks the common assumption regarding resource consumption characteristics. The scheduling criteria is based on the task's resources consumption. KubFBS was tested through extensive experiments using a simulation tool called Node Simulator, which simulates node resources and pod states in Kubernetes. Experimental results demonstrated KubFBS's superiority over baseline methods such as Kube-Scheduler, Optimus [26], Tetris [11], and Kubeshare [43] in terms of load balancing capability and scheduling time.

In May 2022, SchedTune [3], an innovative GPU scheduling platform tailored for heterogeneous deep learning environments, was introduced. Heterogeneous environments refer to setups where the resources, such as GPUs, vary in specifications like memory capacity, computational power, and architecture. SchedTune implements dynamic resource allocation and job placement functionalities within Kubernetes infrastructure to optimize job execution by minimizing communication costs among GPUs. It addresses the complex challenges of scheduling diverse deep learning workloads by leveraging ML-based predictive models to estimate GPU memory requirements accurately. Through extensive experimentation on several deep learning architectures like ResNet [12], DenseNet [14], or VGG [34], SchedTune demonstrates its ability to avoid Out of Memory errors. Compared to the Kube-Scheduler, it improves GPU memory utilization and reduces workload completion time by efficiently matching jobs to appropriate GPUs based on their characteristics. Results highlight SchedTune 's effectiveness in providing fair allocation of cluster-wide GPU time, maintaining progress for all jobs, and efficiently managing GPU heterogeneity.

In September 2023, Voda [13] emerged as an innovative GPU scheduling platform tailored for elastic deep learning environments built on the top of Kubernetes infrastructure. Voda introduces dynamic resource allocation, job placement, and worker migration functionalities to optimize job execution by minimizing communication costs among GPUs. The platform addresses the complex challenges of scheduling diverse deep learning workloads by implementing and comparing multiple scheduling algorithms, including Elastic-FIFO [16], Elastic-Tiresias [42], FfDL Optimizer [15], and AFS-L [47]. Through extensive experimentation and workload simulations, Voda showcases its ability to provide fair allocation of cluster-wide GPU time, maintain progress for all jobs, and efficiently manage GPU heterogeneity. Results demonstrate enhanced fairness, progress, and efficiency in GPU resource utilization, establishing

Voda as a versatile and effective solution for elastic deep learning environments.

7 RESULTS

When it comes to selecting a scheduler, the best option largely depends on specific criteria and requirements, therefore there is no universally optimal choice. Each system has its strengths and weaknesses, and preferences may vary based on the context and goals of the implementation.

Table 1 provides a comparison of schedulers according to their objectives, strategy and specifications. The aim of this table is to help in the selection of a scheduler by having predefined criteria regarding the expectations to be met. This table does not present a quantified comparison of scheduler performance for the different objectives, as there are no benchmarks (see section 8) to provide a common basis for evaluating the metrics. Furthermore, this would require additional work that goes beyond our presentation of the scheduler landscape.

Year	Paper	Goals	Strat.	Specs.	Code
2018	OASiS [4]	🟦 🟥	□	*AR*	-
	Optimus [26]	🟦 🟥	◊	*AFM*	✓
	Gaia [35]	🟦	⊗	*FR*	✓
2019	Kube-Knots [37]	🟦 🟥	◊	*AFM*	✓
	DL^2 [27]	🟥	◊	*AFM*	✓
2020	Gandiva [7]	🟦 🟩	⊙	*AFR*	-
	KubeShare [43]	🟦 🟥 🟨	△ ◊	*AFR*	✓
	Wang et al. [40]	🟦 🟥	◊	*AFR*	-
	HiveD [45]	🟩 🟥	♡ ⊗	*AFR*	✓
2021	Vapor [46]	🟦	△	*AFM*	-
	K-Scheduler [18]	🟥 🟨	△⊙	*AFR*	✓
	KubFBS [20]	🟥 🟨	◊	*AFR*	-
2022	Schedtune [3]	🟦 🟨	◊	*AFM*	-
2023	Voda [13]	🟦 🟨	⊙ ◊	*AFR*	✓

Table 1: Summary of Kubernetes GPU Schedulers.
Goals: 🟦 Utilization 🟩 Fairness 🟥 Job completion time 🟨 Elasticity.
Strategies: □ Exclusive ⊙ Gang Scheduling △ GPU Sharing ◊ Elastic Training ♡ Consolidated Placement ⊗ Topology-Agnostic Placement.
Specifications: *A* **Algorithm** *F* **Framework** *M* **ML-Based** *R* **Rule-based**

This summary inspired us a few remarks. A considerable number of the scheduler proposed have no code available and are therefore more difficult to apprehend. Regarding the distribution of machine-learning or rule-based scheduler, there are no preference among them and are equally distributed. Most of the frameworks comes with there own algorithm. Furthermore, most of the solution we found date of 2022 at the latest. Even though the development of new scheduler continue to arise, these latter are not the subject of published paper.

8 FUTURE DIRECTIONS

The use of GPU resources in clusters is expected to continue to increase due to the growing demand for ML workloads. In addition, the proliferation of new open-source models, such as customized local large language models, makes AI adoption attractive to companies. However, GPU resources are relatively expensive, and their availability could be limited. It is, therefore, fundamental to optimize the use of resources by employing an appropriate scheduler for cluster management. The development and improvement of schedulers are active research areas, attracting considerable interest from companies adopting such solutions.

It would be useful to propose a standardized methodology for evaluating schedulers to facilitate comparison between schedulers and encourage the development of a scientific research community in this field. Such an approach would make it possible to obtain more robust evaluations of schedulers and their performance across a spectrum of measurements. For example, the proposed methodology could encompass benchmarks that simulate real ML workloads, ensuring that schedulers are evaluated under conditions that reflect practical deployment scenarios. In addition, incorporating qualitative assessments, such as feedback from system administrators and developers, could provide insight into the usability and effectiveness of different schedulers in real-world contexts.

Scheduler predictability is another key area of ongoing scientific study. Predictability measures assess the consistency and reliability of scheduler performance under varying workload conditions. A predictable scheduler maintains stable resource allocation patterns and guarantees consistent performance under various workload scenarios. This facet is essential for improving the efficiency and reliability of cluster management systems. Further research in this area will undoubtedly help to advance our understanding of scheduler behavior and inform the design of more robust and adaptable scheduling algorithms. Additionally, studying methods for schedulers to automatically adjust to changing workloads could enhance their ability to predict needs and independently manage resources more effectively in changing conditions.

8.1 Lack of Scientific Publications

By exploring the schedulers available on the internet, we found that there are several solutions other than those mentioned in this paper. However, the development of these has not been accompanied by scientific publications. This raises the question of the gap between the academic world and the open source world.

In fact, many open source solutions are developed without going through the scientific publication process. This can be explained by the fact that the open source community often favours rapid iteration and direct collaboration between developers rather than academic validation. In addition, open source projects are often geared towards the practical resolution of concrete problems, with rapid implementation and immediate feedback from the user community.

However, the absence of scientific publications means that some innovations and techniques developed in open source may not be widely recognised or adopted by the academic community. This separation can limit the spread of knowledge and best practice between these two worlds.

It would therefore be beneficial to find ways of encouraging collaboration between open source developers and academic researchers. This could include initiatives to document and publish the results of open source projects, conferences and workshops that bring these two communities together, or funding programs that support applied research in open source environments. By encouraging these interactions, we could bridge the existing gap and improve the quality and impact of Kubernetes scheduling solutions for GPUs.

9 CONCLUSION

This paper explored the question: "What software methods can be employed to share GPU-constrained resources within Kubernetes environments effectively?". We reviewed and classified numerous papers based on goals, strategies and specifications we established. Our findings underscore that no solution fits all scenarios, the issue presents a multi-objective optimization challenge. Several Pareto-optimal solutions are potentially suitable depending on specific user requirements and constraints.

While various strategies offer viable solutions for sharing GPU resources, selecting the most appropriate approach requires a clear definition of user-specific needs. Future work should focus on creating standardized benchmarks for these strategies, allowing for a more comprehensive comparison and a clearer understanding of their relative effectiveness. Such efforts will enable a more nuanced appreciation of the trade-offs involved, guiding users in navigating the Pareto frontier to find the most effective solutions for their contexts.

ACKNOWLEDGMENTS

We would like to express our gratitude to the reviewers and our teacher for their valuable advice and constructive feedback throughout the research process. Their insights significantly contributed to the improvement of this work.

REFERENCES

[1] [n. d.]. Kubernetes Components. https://kubernetes.io/docs/concepts/overview/components/ Section: docs.

[2] [n. d.]. Production-Grade Container Orchestration. https://kubernetes.io/

[3] Hadeel Albahar, Shruti Dongare, Yanlin Du, Nannan Zhao, Arnab K. Paul, and Ali R. Butt. 2022. SchedTune: A Heterogeneity-Aware GPU Scheduler for Deep Learning. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 695–705. https://doi.org/10.1109/CCGrid54584.2022.00079

[4] Yixin Bao, Yanghua Peng, Chuan Wu, and Zongpeng Li. 2018. Online Job Scheduling in Distributed Machine Learning Clusters. http://arxiv.org/abs/1801.00936 arXiv:1801.00936 [cs].

[5] Carmen Carrión. 2022. Kubernetes scheduling: Taxonomy, ongoing issues and challenges. *Comput. Surveys* 55, 7 (2022), 1–37.

[6] Carmen Carrión. 2022. Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges. *Comput. Surveys* 55, 7 (2022), 138:1–138:37. https://doi.org/10.1145/3539606

[7] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. 2020. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*. ACM, Heraklion Greece, 1–16. https://doi.org/10.1145/3342195.3387555

[8] Hongwen Dai, Zhen Lin, Chao Li, Chen Zhao, Fei Wang, Nanning Zheng, and Huiyang Zhou. 2018. Accelerate GPU Concurrent Kernel Execution by Mitigating Memory Pipeline Stalls. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 208–220. https://doi.org/10.1109/HPCA.2018.00027 ISSN: 2378-203X.

[9] Ghofrane El Haj Ahmed, Felipe Gil-Castiñeira, and Enrique Costa-Montenegro. 2021. KubCG: A dynamic Kubernetes scheduler for heterogeneous clusters. *Software: Practice and Experience* 51, 2 (2021), 213–234. https://doi.org/10.1002/spe.2898 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2898.

[10] Wei Gao, Qinghao Hu, Zhisheng Ye, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. 2022. Deep Learning Workload Scheduling in GPU Datacenters: Taxonomy, Challenges and Vision. http://arxiv.org/abs/2205.11913 arXiv:2205.11913 [cs].

[11] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. *SIGCOMM Comput. Commun. Rev.* 44, 4 (aug 2014), 455–466. https://doi.org/10.1145/2740070.2626334

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]

[13] Tsung-Tso Hsieh and Che-Rung Lee. 2023. Voda: A GPU Scheduling Platform for Elastic Deep Learning in Kubernetes Clusters. In *2023 IEEE International Conference on Cloud Engineering (IC2E)*. 131–140. https://doi.org/10.1109/IC2E59103.2023.00023 ISSN: 2694-0825.

[14] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2018. Densely Connected Convolutional Networks. arXiv:1608.06993 [cs.CV]

[15] Kamesh Jayaram, Archit Verma, Falk Pollok, Rania Khalaf, Vinod Muthusamy, Parijat Dube, Vatche Isahagian, Chen Wang, Benjamin Herta, Scott Boag, Diana Arroyo, and Asser Tantawi. 2019. *FfDL: A Flexible Multi-tenant Deep Learning Platform*. https://doi.org/10.1145/3361525.3361538 Pages: 95.

[16] Jitti Jungwattanakit, Manop Reodecha, Paveena Chaovalitwongse, and Frank Werner. 2009. A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *Computers & Operations Research* 36, 2 (Feb. 2009), 358–378. https://doi.org/10.1016/j.cor.2007.10.004

[17] Alex Kesselman and Yishay Mansour. 2003. Adaptive AIMD congestion control. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing (PODC '03)*. Association for Computing Machinery, New York, NY, USA, 352–359. https://doi.org/10.1145/872035.872089

[18] Sejin Kim and Yoonhee Kim. 2022. K-Scheduler: dynamic intra-SM multitasking management with execution profiles on GPUs. *Cluster Computing* 25 (Feb. 2022), 1–21. https://doi.org/10.1007/s10586-021-03429-7

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (May 2015), 436–444. https://doi.org/10.1038/nature14539

[20] Zijie Liu, Can Chen, Junjiang Li, Yi Cheng, Yingjie Kou, and Dengyin Zhang. 2022. KubFBS: A fine-grained and balance-aware scheduling system for deep learning tasks based on kubernetes. *Concurrency and Computation: Practice and Experience* 34, 11 (2022), e6836. https://doi.org/10.1002/cpe.6836 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6836.

[21] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient GPU Cluster Scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 289–304. https://www.usenix.org/conference/nsdi20/presentation/mahajan

[22] Aidan O Mahony and Emanuel Popovici. 2019. A Systematic Review of Blockchain Hardware Acceleration Architectures. In *2019 30th Irish Signals and Systems Conference (ISSC)*. 1–6. https://doi.org/10.1109/ISSC.2019.8904936

[23] Sparsh Mittal and Jeffrey S Vetter. 2014. A survey of methods for analyzing and improving GPU energy efficiency. *ACM Computing Surveys (CSUR)* 47, 2 (2014), 1–23.

[24] Sparsh Mittal and Jeffrey S Vetter. 2015. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 1–35.

[25] Chang Park, Taekyung Heo, and Jaehyuk Huh. 2016. Efficient Intra-SM Slicing through Dynamic Resource Partitioning for GPU Multiprogramming. 217–229. https://doi.org/10.1109/ISCA.2016.28

[26] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, Porto Portugal, 1–14. https://doi.org/10.1145/3190508.3190517

[27] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, Chen Meng, and Wei Lin. 2019. DL2: A Deep Learning-driven Scheduler for Deep Learning Clusters. https://doi.org/10.48550/arXiv.1909.06040 arXiv:1909.06040 [cs, stat].

[28] Nigel Poulton. 2023. *The kubernetes book*. NIGEL POULTON LTD.

[29] Guillem Pratx and Lei Xing. 2011. GPU computing in medical physics: A review. *Medical Physics* 38, 5 (2011), 2685–2697. https://doi.org/10.1118/1.3578605 arXiv:https://aapm.onlinelibrary.wiley.com/doi/pdf/10.1118/1.3578605

[30] Vignesh T. Ravi, Michela Becchi, Wei Jiang, Gagan Agrawal, and Srimat Chakradhar. 2012. Scheduling Concurrent Applications on a Cluster of CPU-GPU Nodes. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012) (CCGRID '12)*. IEEE Computer Society, USA, 140–147. https://doi.org/10.1109/CCGrid.2012.78

[31] Zeineb Rejiba and Javad Chamanara. 2022. Custom Scheduling in Kubernetes: A Survey on Common Problems and Solution Approaches. *Comput. Surveys* 55, 7 (2022), 151:1–151:37. https://doi.org/10.1145/3544788

[32] Kittisak Sajjapongse, Xiang Wang, and Michela Becchi. 2018. 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 (HPDC '13)*. Association for Computing Machinery, New York, NY, USA, 179–190. https://doi.org/10.1145/2462902.2462911

[33] Souradip Sarkar, Turbo Majumder, Ananth Kalyanaraman, and Partha Pratim Pande. 2010. Hardware accelerators for biocomputing: A survey. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 3789–3792. https://doi.org/10.1109/ISCAS.2010.5537736

[34] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

[35] Shengbo Song, Lelai Deng, Jun Gong, and Hanmei Luo. 2018. Gaia Scheduler: A Kubernetes-Based Scheduler Framework. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. 252–259. https://doi.org/10.1109/BDCloud.2018.00048

[36] John E. Stone, David J. Hardy, Ivan S. Ufimtsev, and Klaus Schulten. 2010. GPU-accelerated molecular modeling coming of age. *Journal of Molecular Graphics and Modelling* 29, 2 (2010), 116–125. https://doi.org/10.1016/j.jmgm.2010.06.010

[37] Prashanth Thinakaran, Jashwant Raj Gunasekaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. 2019. Kube-Knots: Resource Harvesting through Dynamic Container Orchestration in GPU-based Datacenters. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. 1–13. https://doi.org/10.1109/CLUSTER.2019.8891040 ISSN: 2168-9253.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[39] Richard Vuduc, Aparna Chandramowlishwaran, Jee Choi, Murat Guney, and Aashay Shringarpure. 2010. On the limits of GPU acceleration. In *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, Vol. 13.

[40] Shaoqi Wang, Oscar J. Gonzalez, Xiaobo Zhou, Thomas Williams, Brian D. Friedman, Martin Havemann, and Thomas Woo. 2020. An Efficient and Non-Intrusive GPU Scheduling Framework for Deep Learning Training Systems. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Atlanta, GA, USA, 1–13. https://doi.org/10.1109/SC41405.2020.00094

[41] Zhenning Wang, Jun Yang, Rami Melhem, Bruce Childers, Youtao Zhang, and Minyi Guo. 2016. Simultaneous multikernel GPU: Multi-tasking throughput processors via fine-grained sharing. In *2016 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 358–369. https://ieeexplore.ieee.org/abstract/document/7446078/

[42] Yidi Wu, Kaihao Ma, Xiao Yan, Zhi Liu, and James Cheng. 2019. *Elastic deep learning in multi-tenant GPU cluster.*

[43] Ting-An Yeh, Hung-Hsin Chen, and Jerry Chou. 2020. KubeShare: A Framework to Manage GPUs as First-Class and Shared Resources in Container Cloud. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, Stockholm Sweden, 173–184. https://doi.org/10.1145/3369583.3392679

[44] Peifeng Yu and Mosharaf Chowdhury. [n. d.]. Salus: Fine-Grained GPU Sharing Primitives for Deep Learning Applications. ([n. d.]).

[45] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Fan Yang, Lidong Zhou, Mao Yang, Francis C. M. Lau, Yuqi Wang, Yifan Xiong, and Bin Wang. 2020. {HiveD}: Sharing a {GPU} Cluster for Deep Learning with Guarantees. 515–532. https://www.usenix.org/conference/osdi20/presentation/zhao-hanyu

[46] Xiaorui Zhu, Lei Gong, Zongwei Zhu, and Xuehai Zhou. 2021. Vapor: A GPU Sharing Scheduler with Communication and Computation Pipeline for Distributed Deep Learning. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, New York City, NY, USA, 108–116. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00028

[47] Sonia Zouaoui, Lotfi Boussaid, and Mtibaa Abdellatif. 2016. *CPU scheduling algorithms: Case & comparative study.* https://doi.org/10.1109/STA.2016.7951997 Pages: 164.

[48] 김세진, 진계신, 염헌영, and 김윤희. 2021. GPU의 효율적인 자원 활용을 위한 동시 멀티태스킹 성능 분석. *정보과학회논문지* 48, 6 (June 2021), 604–611. https://doi.org/10.5626/JOK.2021.48.6.604

Received 7 May 2024; revised 21 May 2024; accepted 21 May 2024