National College of Ireland

Optimizing GPU Resource Allocation and Scheduling using a Hybrid Scheduler

MSc Research Project
Cloud Computing

Sai Nitish Kavali
Student ID: 22125809

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sai Nitish Kavali
Student ID:	22125809
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	14/12/2023
Project Title:	Optimizing GPU Resource Allocation and Scheduling using a Hybrid Scheduler
Word Count:	5281
Page Count:	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Nitish Kavali
Date:	29th January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimizing GPU Resource Allocation and Scheduling using a Hybrid Scheduler

Sai Nitish Kavali
22125809

Abstract

This research explores the evolving landscape of computational technology, focusing on the critical role of Graphics Processing Units (GPUs) in addressing complex computing problems. Initially designed for gaming and visualization, GPUs have transformed into powerful instruments, driving advancements in deep learning, scientific simulations, and video rendering. Despite their advantages, limited studies have delved into GPU scheduling in cloud environments, posing challenges for organizations managing GPU-intensive workloads. This research introduces a hybrid scheduler, combining FIFO and Availability-based methodologies for efficient GPU resource utilization. The scheduler's objective is to maximize throughput and minimize resource wastage by orchestrating task execution across homogeneous GPUs. The study specifically investigates the application of this hybrid scheduler in training Convolutional Neural Network (CNN) algorithms, demonstrating its effectiveness in optimizing resource utilization and adapting to dynamic machine learning workloads. The research addresses the underutilization of GPUs by introducing allocation methodologies based on availability and FIFO, and other strategies, enhancing overall hardware resource utilization.

Keywords: **GPU Scheduling, Hybrid scheduler, FIFO, Availability based Scheduling, Homogeneous GPUs**

1 Introduction

In times of the ever-evolving landscape of computational technology, the Graphical Processing Unit (GPU) has emerged as a vital component, thereby altering how difficult computer problems are tackled. (GPUs) were initially developed to produce visuals for gaming and visualization. However, they have since evolved into powerful instruments that can be used for a variety of purposes, such as deep learning, scientific simulations, and video renderingIserte et al. (2019). Graphics processing units have swiftly developed into massively parallel computing systems that are capable of simultaneously executing thousands of threads for computer programs. Because of their raw processing power, they have been adopted in a variety of high-performance computing fields, such as scientific workloads, artificial intelligence and machine learning, analytics, and more.

Despite having several advantages the studies related to GPU scheduling in the cloud environment are limited. The demand for GPU-intensive workloads has been steadily increasing, and as a result, organizations are facing difficulties in effectively managing

and utilizing resources such as GPU. Graphics processing units (GPUs) are among the devices that are cost-effective. However, their availability is frequently restricted, and the expenses associated with maintaining GPU clusters can be prohibitive for many businesses.

A scheduler is an essential component that is responsible for managing the execution of tasks or processes on a GPU. In the context of computing, a scheduler is a vital component. To guarantee that available resources are utilized effectively and efficiently, it distributes resources. The consistency, responsiveness, and overall performance of the system are all maintained by schedulers, who play a crucial part in this process.

All modern GPU drivers provide mechanisms for developers to control and optimize task scheduling on GPUs for specific applications but is less efficient when compared to a hybrid scheduler. A hybrid scheduler represents an innovative approach to the task and resource management by combining multiple scheduling techniques to address the complexities of modern computing environments. In this context, the Hybrid scheduler uses both FIFO along with the availability Scheduler which will enable the scheduling by using first in first out as its primary checking algorithm and then based on the availability of the resources. The primary objective of GPU scheduling methods is to facilitate the efficient utilization of GPU resources Liu et al. (2016). This involves orchestrating the execution of tasks across different GPUs in a manner that maximizes overall throughput and minimizes resource wastage.

Their capability extends to optimizing resource utilization, managing task scheduling, and adjusting to the dynamic nature of machine learning workloads when hybrid schedulers are used in the training of Convolutional Neural Network (CNN) algorithms Xue et al. (2022). This is because hybrid schedulers can maximize the utilization of resources. In the past few years, there has been a discernible pattern that pertains to the heterogeneity of resources in the servers, such as the presence of various types of graphics processing units (GPUs) in the cluster. In the same way, there has been an increase in the utilization of GPUs in enterprise server systems, for the purpose of carrying out computationally intensive tasks such as the rendering of videos or the processing of images.

Because graphics processing units are not considered to be first-class schedulable entities, GPUs are viewed as the target devices that are chosen by the applications that are currently running Yeung et al. (2020). This results in underutilization, in which some devices are over-utilized while others are left idle. Also serial execution of GPU tasks decreases the performance of the resource which otherwise can be achieved by task parallelizing. Consequently, to enhance the utilization of the graphics processing unit (GPU), certain methods have already been established. These include allocation methodologies that are utilized based on availability, Least attended service, FIFO, and others. These methodologies support the enhancement of the utilization of the resources in the hardware that underlies the GPUGao et al. (2022).

Research Question: How can we achieve a state where GPUs are effectively scheduled and overloads are maintained to be minimum even when QOS is achieved?

Integrating FIFO and Availability-based methods to find a balance between orderly task execution and optimal resource usage is what makes this work stand out. Real-world applications are emphasized, and the scheduler is especially used to train Convolutional Neural Network (CNN) algorithms, aiming to solve the problems that come up with changing machine learning workloads. Specifically, the study aims to reduce GPU underutilization by introducing allocation methods based on availability and FIFO, while also improving the total use of hardware resources.

2 Related Work

GPUs have become appealing entities in the world of parallel computing due to their unique computational capabilities. They promise increased performance across a variety of domains for a variety of applications. As stated GPUs not being considered as first class entities and task selecting the GPU at runtime is leading to the underutilization of the resource. There are some studies done on this and I will discuss clearly about them in this section.

2.1 GPU Co-Scheduling and parallelism

Embedded systems that are enhanced with graphics processing units (GPUs) have become increasingly common in safety-critical real-time systems such as driverless automobiles. On the other hand, these embedded GPUs frequently function under computational limits not just because of limitations in terms of size, weight, and power (SWaP), but also because of cost constraints. It's clear that co-scheduling is an essential tactic for achieving maximum performance in this setting. By allowing different applications to utilize a graphics processing unit (GPU) simultaneously, co-scheduling helps to maximize the GPU's utilization. When it comes to platforms with limited computing capabilities, such as the NVIDIA Jetson TX1, this paradigm is particularly significant.

Authors Otterness et al. (2017) to investigate the inner workings of GPU co-scheduling employed a benchmarking framework. Additionally, this framework makes it possible to conduct an in-depth investigation into the internal scheduling policies that are utilized by the black-box hardware and software setups that make GPU co-scheduling possible. Researchers intend to provide useful insights into the efficiency and limitations of co-scheduling algorithms on embedded GPUs by assessing the performance of numerous apps that are simultaneously accessing a GPU while simultaneously measuring the performance of the GPU.

Similarly to this Authors Agarwal et al. (2015) detailed how the introduction of CUDA, which is NVIDIA's parallel computing architecture, has been a game-changer in terms of completely harnessing the enormous potential that GPUs possess. By taking advantage of the inherent parallelism that GPUs possess, CUDA makes it possible to achieve a significant improvement in computing performance. The execution of sophisticated algorithms on graphics processing units (GPUs) has been made possible as a result of this architectural shift, which promises major gains in computational efficiencyRanganath et al. (2021).

2.2 GPU-Driven Real-Time Deep Learning Optimization

The challenge of GPU kernel launch overhead in real-time inference on contemporary mobile GPUs is addressed by the authors Kim et al. (2021). A comprehensive investigation on the inference time for regularly employed CNNs is carried out by them, which reveals a significant amount of overhead that can be attributed to GPU kernel launches. A performance model is developed in this work, which also identifies the components that are responsible for this overhead. This model makes a prediction about the best time to flush the kernel, which results in a low amount of overhead. The results of the experiments show that the inference of different CNNs on Adreno 650 GPU and Mali G76 GPU may be significantly sped up by up to 64 percent and 31 percent, respectively, when TensorFlow Lite and ARM Compute Library are utilized.

A significant amount of scheduling overhead and excessive serial execution are common problems that plague existing deep-learning frameworks. In order to address these concerns authors Kwon et al. (2020), employed a revolutionary technique in Nimble known as ahead-of-time (AoT) scheduling. This technique ensures that the scheduling procedure is completed prior to the execution of GPU kernels, hence lowering the amount of runtime scheduling overhead. Nimble additionally makes use of numerous GPU streams in order to automatically parallelize processes that are performed on the GPU. When compared to PyTorch and other cutting-edge inference systems, such as TensorRT and TVM, the evaluation reveals substantial speedups, with inference speeds reaching up to 22.34 times faster and training speeds reaching 3.61 times faster. By a margin of up to 2.81 times and 1.70 times, respectively, Nimble outperforms its competition.

2.3 Efficient GPU Task Management

When it comes to the ever-expanding market for set-top and portable devices, the demand for increased performance despite limited resources is what drives the incorporation of Graphics Processing Units (GPUs). Authors Adriaens et al. (2012) have implemented a new novel multitasking technique, It is called "Spatial Multitasking." Spatial multitasking, in contrast to more conventional techniques of dividing up GPU time among apps, enables the simultaneous division of GPU resources among a number of different applications. Applications that use general-purpose graphics processing units (GPGPUs) are not being used to their full potential, which suggests that spatial multitasking can greatly improve performance. By means of simulation, the research illustrates that cooperative multitasking can achieve an average speed of up to 1.19, highlighting the potential efficiency improvements that might occur when many applications share the graphics processing unit (GPU).

While the authors Ausavarungnirun et al. (2018) addressed the challenges in multi-application concurrency. The difficulties associated with running many applications concurrently on GPUs are discussed, with an emphasis placed on the constraints imposed by virtual memory support. The majority of the time, many applications will share a single graphics processing unit (GPU) in large-scale computing systems; yet, contemporary GPUs do not provide comprehensive support for effective multi-application concurrency.

The authors Ausavarungnirun et al. (2018) conduct an analysis of the performance overheads and identify virtual memory methods as a significant barrier. In particular,

they point out that the performance of address translation is disappointing. In this paper, they provide MASK, a unique GPU system that incorporates address-translation-aware cache and memory management methods. MASK significantly reduces the overhead of address translation, resulting in increases in system throughput of 57.8%, IPC throughput of 43.4%, and application-level unfairness of 22.4%. The speed of MASK is very close to that of an ideal GPU system, and it does not have any address translation cost. This represents a substantial leap in the way that GPU multitasking difficulties are currently being addressed.

Authors Albahar et al. (2022) introduced a novel method that makes use of machine learning (ML) to forecast the amount of graphics processing unit (GPU) memory that will be required for a particular position. By taking this method, it is possible to reduce the occurrence of OOM issues and enhance the GPU's overall performance. Even though the TLB-based technique is more efficient which was quoted by Ausavarungnirun et al. (2018), it still has the potential to cause performance issues when numerous applications are operating at the same time. On the other hand, the method based on machine learning has the potential to be more accurate and to deliver much more efficient performance.

2.4 GPU Optimization for Efficiency

The Authors Hong et al. (2017) focuses on GPU virtualization in the context of heterogeneous computing for cloud platforms, this paper Mei et al. (2017) delves into energy conservation in CPU-GPU hybrid clusters. In both cases, the authors recognize the critical significance of optimizing resource usage to achieve specific goals. where Hong et al. (2017) aims to reduce operational costs and enhance resource and energy efficiency in cloud computing through GPU virtualization, while Mei et al. (2017) targets energy conservation in large data centers, where even a small reduction in consumption can result in substantial cost savings. Both papers employ optimization techniques and algorithms to achieve their objectives, showcasing a commitment to improving the sustainability and performance of contemporary computing systems.

2.5 Scheduling Algorithms

While many GPU scheduling algorithms are present, there isn't much research on heterogeneous GPU resource scheduling in a cluster for multitenant environments. Author Sengupta et al. (2013) discussed scheduling policies, including FIFO, which is easy to implement but does not account for GPU characteristics.

Gmin improves performance by assigning tasks to GPUs that have lower loads, yet ignores cluster heterogeneity. Least attained service (LAS) reduces application wait time for GPU services, while True Fair share (TFS) ensures equitable GPU resource distribution. All these policies ignored cluster heterogeneity. In contrast, Narayanan et al. (2020) explained how considering Executing a job on a K80 GPU and another on a V100 will result in equal resource usage but lower performance due to the K80 GPU's performance.

Narayanan et al. (2020) proposed an optimization problem for Max-Min fairness, maximizing the minimum effective throughput of all users while avoiding over-provisioning the cluster. Also, finish-time fairness is the ratio of job completion time using the given

allocation to the period of time using 1/n of the cluster. Both schedule jobs in a diverse cluster to ensure fair resource allocation.

3 Methodology

Firstly, a systematic analysis of GPU architectures, capabilities, and limitations is conducted. This analysis serves to establish a comprehensive understanding of the technological landscape, specifically with regard to how existing GPU scheduling methodologies can be adapted and improved for machine learning tasks.

During the design phase of the Hybrid Scheduler, certain needs from the proposal are taken into account. These include adding a machine learning module that makes it easy to use GPUs to simulate different scheduling scenarios and a front-end interface made with the Flask Framework that lets us see how tasks are being assigned and scheduled in real time on the GPUs. The design uses EC2 instance, g4dn.12xlarge, which is running Windows Server and has 4 NVIDIA Tesla GPUs and 48 virtual CPUs.

The next step in the implementation process is to set up the Hybrid Scheduler on the chosen EC2 instance. The machine learning module is built in to simulate GPU usage during scheduling, and the Flask Framework-based front end is made to make it easy to keep updated on scheduling activities in real-time.

Representative Convolutional Neural Network (CNN) algorithms are selected for training to simulate realistic machine learning workloads. The machine learning module is executed to invoke GPUs, and the impact on GPU utilization during CNN training is observed and analyzed.

The experimental setup includes Windows Server, 48 vCPUs, and 4 NVIDIA Tesla GPUs. The specifications of the EC2 instance are listed so that the configuration is clear. During simulated task execution, different terminals are used to put loads on GPUs in order to test the Hybrid Scheduler's response and ability to assign tasks efficiently.

- EC2 Instance: An EC2 instance refers to a virtual server that is hosted on the Amazon Web Services (AWS) platform. It offers the computational framework for deploying and executing the Hybrid Scheduler. The given instance type, such as g4dn.12xlarge, dictates the computational resources that are accessible, including vCPUs and GPUs.

- Code Editor: The code editor is a software tool that provides a workspace for writing and managing the source code of the Hybrid Scheduler within the EC2 instance. The user can choose any integrated development environment (IDE) or a basic text editor to generate scripts and programs for the machine learning module, scheduling logic, and Flask web application.

- Terminal: The Terminal is the graphical user interface that allows users to interact with the EC2 instance through command-line commands. The Hybrid Scheduler use it as the interface for initiating, regulating, and monitoring its many components.

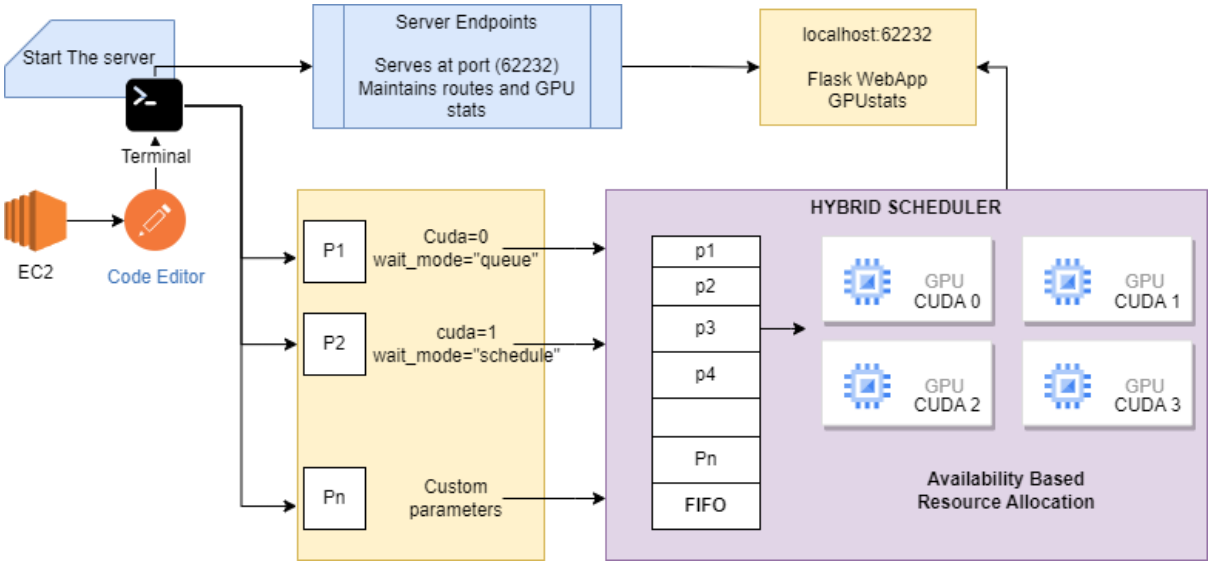

Figure 1: WorkFlow of the Project

- Server Endpoints: The machine learning module and Flask web application provide access points on the server. These endpoints refer to certain URLs or routes that are responsible for handling requests. As an illustration, the machine learning module may have an endpoint such as _is_totally_free, whilst the Flask web app could have endpoints like /gpus/¡gpu$_ids > toreceiveinformationaboutscheduling$.

The Flask web application is a Python web framework used to construct the graphical user interface of the Hybrid Scheduler. The web app is hosted locally on the EC2 instance and can be accessed by developers using the web browser using the address localhost. The online application offers a real-time visual depiction of scheduling operations, task allocation, and GPU utilization.

The Hybrid Scheduler is the central element that oversees the distribution of resources, specifically GPU resources, according to their availability. The system incorporates a scheduling algorithm that adaptively modifies the distribution of jobs to maximize the efficiency of GPU utilization. This may entail the prioritization of jobs, the management of queues, and the assurance of optimal resource utilization.

Flow of Operations: Code editor is used to generate scripts and programs for the machine learning module. This module is intended to trigger GPU activities, such as training Convolutional Neural Networks (CNNs), by exposing designated server endpoints. The Terminal is employed to execute and oversee the Hybrid Scheduler, enabling developers to initiate, terminate, and supervise the scheduler's functionality. Additionally, it functions as a means of engaging with both the Flask web application and the machine learning module. The server endpoints offered by the machine learning module and Flask web application serve as channels for communication. When accessed, various endpoints initiate distinct capabilities, such as commencing GPU training or presenting schedule information. The Flask web application, operating on the local machine, functions as the graphical user interface. It establishes communication with the machine learning module and the Hybrid Scheduler in order to retrieve and present up-to-date data on scheduling

operations, task allocation, and GPU utilization.

The Hybrid Scheduler utilizes an available-based resource allocation technique to dynamically modify resource allocation according to the availability of GPU resources. The Flask web application sends requests to it, and it efficiently distributes duties across the system.

4 Design Specification

To Implement the proposed hybrid scheduler the requirements or the specifications needed are as follows:

4.1 Design Requirements

The basic requirements for this proposed Hybrid scheduler are

- Machine learning module: One of the most important parts of the hybrid scheduler is the machine learning module. Its main function is to communicate with and activate the GPUs (Graphics Processing Units). It tries to replicate real-world situations where GPUs are used for machine learning tasks by doing this. In particular, when the module is called, it is intended to start Convolutional Neural Networks (CNNs) in training. This is a crucial step in order to simulate the scheduling scenarios that the Hybrid Scheduler will oversee by placing a workload on the GPUs.

- Flask Framework: To generate an interface that is easy to use, the Hybrid Scheduler front end makes use of the Flask Framework. Here, the graphical user interface (GUI) that gives the GPUs real-time scheduling and task allocation data is developed using Flask. Through interaction with this front-end interface, users can track GPU utilization and learn how the scheduler distributes tasks among the available GPUs.

4.1.1 Instance Specifiations

- An Amazon Elastic Compute Cloud (EC2) instance is selected to serve as the environment for simulating the Hybrid Scheduler implementation. The specifications of this EC2 instance are as follows:
  - Instance Type: g4dn.12xlarge
  - Windows Server
  - vCPU : 48
  - GPU : 4 NVIDIA Tesla GPUs

The g4dn.12xlarge instance type, which is designed for GPU-intensive workloads and machine learning, guarantees that the environment closely complies with the Hybrid Scheduler's specifications. The computational power required for GPU-centric tasks is provided by the four NVIDIA Tesla GPUs included in the package.

4.2 Proposed design of Hybrid Scheduler

As discussed in Section 3, The hybrid scheduler is designed and implemented as follows:

1. The Hybrid Scheduler calls upon the GPUs to begin its work. The machine learning module's machine learning model, which is used to accomplish this, is run. Convolutional Neural Networks (CNNs) are trained in this context by the ML model, which is specifically made to cause GPU activity. This step maximizes GPU utilization by actively putting the GPUs to work on a computationally demanding task, while also simulating the workload that the scheduler will oversee.

2. The scheduler loads the GPUs to simulate multiple tasks once they are being used actively. These computer-generated tasks replicate real-world situations in which several processes or calculations compete for GPU resources. The objective is to evaluate how well the Hybrid Scheduler handles resource allocation, GPU utilization optimization, and these concurrent tasks.

3. The application's home screen dynamically displays the status and outcomes of task allocation, scheduling, and GPU utilization. The server.py file can be used by the user to launch the application and begin this monitoring. It can be seen how the Hybrid Scheduler distributes tasks and maximizes resource utilization among the available GPUs by accessing real-time insights into GPU utilization from the home screen.

5 Implementation

The development of the hybrid scheduler for GPU resource utilization is a rigorous procedure that involves translating theoretical principles into concrete actions.

1. Environment Setup: During the initial stage, Selecting a cloud environment that matches the project's GPU resource needs is a main task. By selecting AWS, a specific EC2 instance is set up, with careful consideration given to choose an instance type such as g4dn.12xlarge that can support NVIDIA GPUs. The configuration encompasses the establishment of security groups to efficiently control incoming and outgoing traffic.

2. Development of Hybrid Scheduler: Python, a flexible programming language renowned for its ability to seamlessly interact with machine learning frameworks and GPU libraries. PyTorch is utilized to optimize GPU computations. The code is designed to be modular and scalable, combining the logic of FIFO and Availability-based approaches.

3. Integration of FIFO and Availability-Based Strategies: The fusion of FIFO and Availability-based techniques constitutes the fundamental basis of the scheduler's logic. Algorithms are meticulously designed to rank activities according to their attributes and dynamically distribute GPU resources based on their availability. Ensuring that there is a balance of organized implementation and optimal utilization of resources becomes of utmost importance.

4. Maximizing Throughput and Minimizing Resource Wastage: The scheduler's algorithms are optimized to increase throughput by giving priority to tasks with high computing requirements. Load balancing strategies are employed to provide a fair allocation of tasks among the GPUs that are currently accessible. The primary objective is to reduce the inefficient use of resources by making intelligent decisions.

5. Application to CNN Training: Incorporating the hybrid scheduler with a specialized module for training Convolutional Neural Network (CNN) algorithms as they delve into the realm of machine learning. This integration represents a significant milestone, as it brings the scheduler in line with the real requirements of deep learning and allows it to adjust dynamically to changing machine learning workloads. It is mainly used to keep the load active on GPUs.

6. Performance Metrics and Benchmarks: Establishing crucial performance measures, such as GPU use, as the basis for complete benchmarks. Monitoring tools and recording systems are utilized to obtain comprehensive performance data throughout tests. The objective is to provide quantifiable metrics for assessing the efficacy of the hybrid scheduler.

7. Data Collection and Analysis: Gathering data on the scheduler's performance by conducting a series of trials. The monitoring of GPU use and task execution times is conducted with great scrutiny. An in-depth analysis is conducted using the defined metrics and benchmarks, providing insights into the strengths, flaws, and potential areas for growth.

8. Optimization and Further Strategies: Refining the implementation based on the analytical results, with a specific emphasis on optimization. The algorithms used in the scheduler are optimized to improve efficiency, and the exploration also encompasses new methods beyond availability and FIFO. Further enhancement is being pursued by evaluating the factors related to workload characteristics in order to make decisions on dynamic resource allocation.

9. Validation and Future Work: Validation serves as the ultimate point of evaluation. This document presents an analysis of the practical constraints faced during the implementation process, and suggests potential directions for further research.

6 Evaluation

In this section, I will discuss the results and outcomes of the experiments that I performed and the implications that I faced in the research process of implementing this hybrid scheduler. So, As discussed in the above sections, the hybrid scheduler works by scheduling the tasks to the GPU based on two criteria first, the FIFO and then based on the availability. While implementing this hybrid scheduler and evaluating its performance, the below experiments are conducted.

6.1 Multi tenant Environment Scheduling

Initially, I tried to implement the hybrid scheduler in a multitenant environment wherein multiple tenants try to request the same GPU resources, and the hybrid scheduler schedules the tasks based on the conditions and the needs of the tenants. In this process to

achieve multi-tenancy I created two instances and tried to allocate the same GPUs to both instances and it was not successful as the GPUs cannot be shared between the resources in AWS. Wherein the requested GPUs will be allocated to only one EC2 instance at a particular time and the GPU in two different servers are different. So the simulation of a multitenant environment failed and as an alternative I started with the second experiment where in the single EC2 instance is allocated with the multiple GPUs and the hybrid scheduler will schedule the tasks based on the conditions and the utilization of the GPU and the availability of the GPU.

6.2 Queue for a Requested Resource

After the attempt to create the multitenant environment was unsuccessful, I started implementing the hybrid scheduler to the single instance with multiple GPUs. For this, I have created an instance type of g4dn.12xlarge which has 4 GPUs that can be used to schedule the tasks running. As discussed in the methodology, I made use of the deep learning Convolutional neural networks, which help us keep the GPUs invoked to check for the scheduling done by the scheduler that we have developed. In this experiment, I simulated some load onto the GPUs to keep GPU 1 busy and later checked for the scheduling of how the next task with no preference of GPU is being scheduled. on the available GPUs.

The experiment goes this way, when the task is assigned to GPU 0, by specifying the preferred GPU and the training is started, the GPU utilization is increased, which can be seen in the UI that displays all the GPU states and utilization. The second task will be allocated to a different GPU, as expected. When the same scenario is repeated but with the parameters of –wait-mode==”queue”, the GPU is waiting for the task to be completed, or, as shown in the figure 2, will be allocated to the same GPU to run it in parallel if the memory requirements as specified are sufficient for the task to be executed.

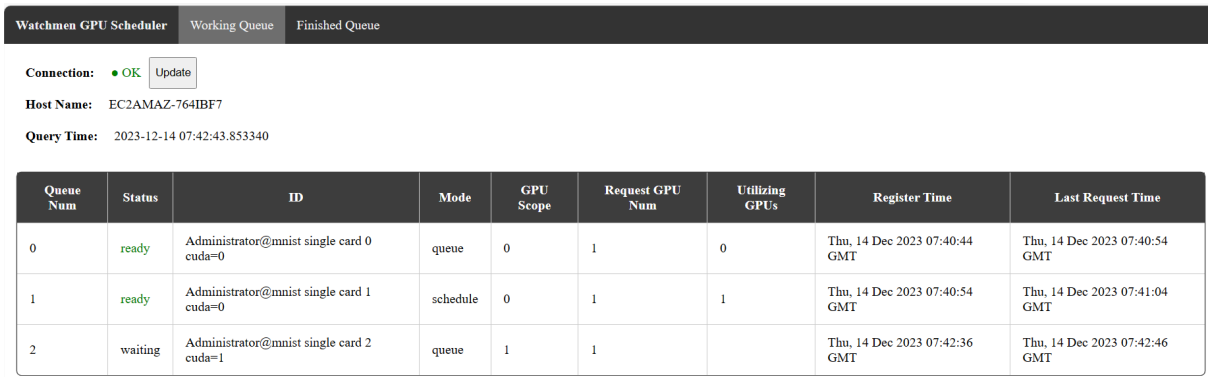

Figure 2: Queuing for requested resource

6.3 Forced Scheduling

Next, after the queuing of tasks was successful, I moved on to testing the implementation of scheduling the tasks to other GPUs if the requested GPU is busy. To achieve this

scenario, I used the parameter –wait-mode again with the value "schedule," which indicates the scheduler to schedule the task to any other available GPU present in the stack. When the tenant requests a resource and the allocated resource is busy and cannot be assigned any other tasks because the memory is over the threshold or the number of tasks running in the resource has reached the limit, then it will be automatically scheduled to another GPU, ignoring the requested GPU. This indicates the tenant is okay with any GPU resource, but the preferred resource is different.

For the experiment, I initially started the three tasks and scheduled them over one GPU, and when the next task requested the same GPU, the GPU threshold was reached, so it would not be allocated to the new task even when the –cuda=0, which indicates GPU 0, but as shown in the figure 3 it will be allocated to another available GPU.

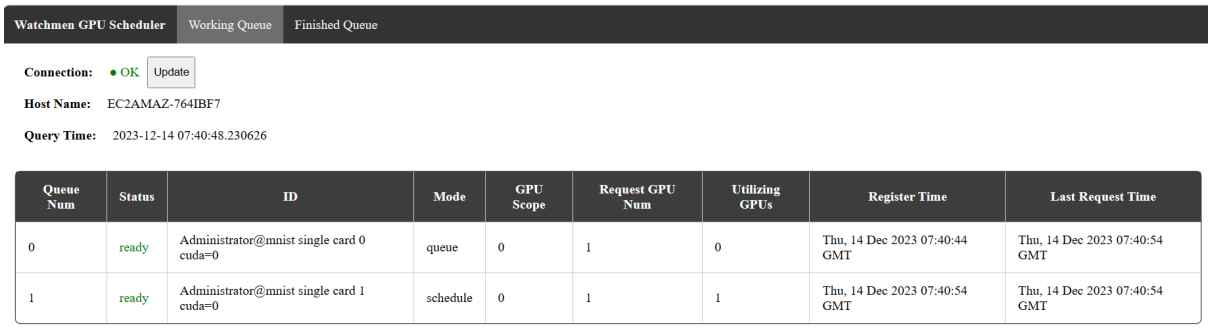

Figure 3: Forced Resource Allocation

6.4 Discussion

The experiments that were carried out in order to evaluate the hybrid scheduler brought to light a number of significant observations. It was seen that the use of the GPU increased significantly while the job was being executed, which is an indication that the scheduler was successful in making the most of the GPU resources. The scheduler displayed an impressive level of scheduling flexibility by dynamically assigning jobs according to the availability of GPUs, the memory requirements, and the preferences of tenants.

Specifically, this adaptability was notably evident in situations such as queuing for requested resources, in which activities were managed methodically in order to maximize GPU use in an effective manner. In addition, the forced scheduling function proved to be useful, as it allowed the scheduler to easily move jobs to alternate GPUs in the event that the desired GPU was already in use or exceeded the thresholds that had been established.

These results collectively demonstrate that the hybrid scheduler is capable of optimizing the management of GPU resources, adapting to a variety of scheduling scenarios, and improving the overall efficiency of the system by intelligently assigning tasks depending on the parameters that have been provided. When it comes to handling the challenges of GPU resource scheduling, the hybrid scheduler has proven to be both strong and versatile, as evidenced by its successful implementation and performance across a variety of scenarios.

7 Conclusion and Future Work

According to the research question, How can we achieve a state where GPUs are effectively scheduled and overloads are maintained to be minimum even when QOS is achieved? Testing the hybrid scheduler revealed its GPU resource management effectiveness. GPU utilization increased significantly during task execution, indicating that the scheduler maximized GPU resources. Flexible scheduling and dynamic job assignment based on GPU availability, memory requirements, and tenant preferences show the hybrid scheduler's adaptability.

In particular, the scheduler queued resources to maximize GPU utilization when the tenant required only that particular GPU. The forced scheduling function allowed the scheduler to seamlessly assign jobs to alternate GPUs when the desired GPU was busy or exceeded thresholds. These results demonstrate the hybrid scheduler's GPU resource management optimization and system efficiency. Hybrid scheduler's effectiveness and versatility in GPU resource scheduling have been shown throughout the experiments. Its robust implementation and performance in various scenarios show its potential as a dynamic and efficient GPU task allocation solution.

While the hybrid scheduler has shown some promising results, there are still areas where there is scope for future work. The parameters that I used in these experiments were meant to be fixed, but they are changed when there are some different requirements, and they can be fine-tuned in the aspects of memory thresholds, job priorities, and scheduling policies. Integrating Machine learning as a part of scheduling, not only for invoking the GPU, can increase the efficiency of the hybrid scheduler and also for improving efficiency in specialized domains, one can customize the scheduler for machine learning training instead of the availability and test the scheduler's performance and scalability as GPUs, tasks, and tenants increase.

References

Adriaens, J., Compton, K., Kim, N. and Schulte, M. (2012). The case for gpgpu spatial multitasking, pp. 1–12.

Agarwal, N., Goyal, A., Maheshwari, G. and Dugtal, A. (2015). Parallel implementation of scheduling algorithms on gpu using cuda, *International Journal of Computer Applications* **127**: 44–49.
URL: *https://api.semanticscholar.org/CorpusID:17533959*

Albahar, H., Dongare, S., Du, Y., Zhao, N., Paul, A. K. and Butt, A. R. (2022). Schedtune: A heterogeneity-aware gpu scheduler for deep learning, *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* pp. 695–705.
URL: *https://api.semanticscholar.org/CorpusID:248942425*

Ausavarungnirun, R., Miller, V., Landgraf, J., Ghose, S., Gandhi, J., Jog, A., Rossbach, C. J. and Mutlu, O. (2018). Mask: Redesigning the gpu memory hierarchy to support multi-application concurrency, *SIGPLAN Not.* **53**(2): 503–518.
URL: *https://doi.org/10.1145/3296957.3173169*

Gao, W., Hu, Q., Ye, Z., Sun, P., Wang, X., Luo, Y., Zhang, T. and Wen, Y. (2022). Deep learning workload scheduling in GPU datacenters: Taxonomy, challenges and vision, *CoRR* **abs/2205.11913**.
URL: *https://doi.org/10.48550/arXiv.2205.11913*

Hong, C.-H., Spence, I. and Nikolopoulos, D. S. (2017). Gpu virtualization and scheduling methods: A comprehensive survey, *ACM Comput. Surv.* **50**(3).
URL: *https://doi.org/10.1145/3068281*

Iserte, S., Prades, J., Reaño, C. and Silla, F. (2019). Improving the management efficiency of gpu workloads in data centers through gpu virtualization, *Concurrency and Computation: Practice and Experience* **33**: e5275.

Kim, S., Oh, S. and Yi, Y. (2021). Minimizing gpu kernel launch overhead in deep learning inference on mobile gpus, *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, HotMobile '21, Association for Computing Machinery, New York, NY, USA, p. 57–63.
URL: *https://doi.org/10.1145/3446382.3448606*

Kwon, W., Yu, G.-I., Jeong, E. and Chun, B.-G. (2020). Nimble: Lightweight and parallel gpu task scheduling for deep learning, *in* H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin (eds), *Advances in Neural Information Processing Systems*, Vol. 33, Curran Associates, Inc., pp. 8343–8354.

Liu, B., Chen, Q., Li, J. and Gao, L. (2016). Ai bcs: A gpu cluster scheduling optimization based on ske model, *Microprocessors and Microsystems* **47**: 121–132.
URL: *https://www.sciencedirect.com/science/article/pii/S014193311630045X*

Mei, X., Chu, X., Liu, H., Leung, Y.-W. and Li, Z. (2017). Energy efficient real-time task scheduling on cpu-gpu hybrid clusters, *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9.

Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A. and Zaharia, M. (2020). Heterogeneity-Aware cluster scheduling policies for deep learning workloads, *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, USENIX Association, pp. 481–498.
URL: *https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak*

Otterness, N., Yang, M., Amert, T., Anderson, J. and Smith, F. D. (2017). Inferring the scheduling policies of an embedded cuda gpu.

Ranganath, K., Suetterlein, J. D., Manzano, J. B., Song, S. L. and Wong, D. (2021). Mapa: Multi-accelerator pattern allocation policy for multi-tenant gpu servers, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, Association for Computing Machinery, New York, NY, USA.
URL: *https://doi.org/10.1145/3458817.3480853*

Sengupta, D., Belapure, R. and Schwan, K. (2013). Multi-tenancy on gpgpu-based servers, pp. 3–10.

Xue, F., Hai, Q., Dong, T., Cui, Z. and Gong, Y. (2022). A deep reinforcement learning based hybrid algorithm for efficient resource scheduling in edge computing environment, *Information Sciences* **608**: 362–374.
URL: *https://www.sciencedirect.com/science/article/pii/S0020025522006727*

Yeung, G., Borowiec, D., Friday, A., Harper, R. and Garraghan, P. (2020). Towards GPU utilization prediction for cloud deep learning, *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, USENIX Association.
URL: *https://www.usenix.org/conference/hotcloud20/presentation/yeung*