Fair and Efficient Resource Scheduling in Heterogeneous Multi-Agent Systems

by

Mohammad Hadi Omidi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The performance of machine-learning applications heavily relies on the choice of the underlying hardware architecture, encompassing factors such as computational power, scalability, memory, and storage capabilities. These hardware choices significantly impact the efficiency and effectiveness of machine-learning systems. Resource-intensive programs can lead to competition for system resources, causing delays, while inefficient resource usage can saturate resources and harm user experience. To address resource variation among applications, resource sharing is implemented, allowing applications to dynamically allocate resources as needed, promoting efficient resource utilization. However, resource-allocation strategies often prioritize performance, potentially overlooking fairness among users or applications, especially in shared environments. Balancing performance optimization and fair resource-allocation is a complex challenge, requiring mechanisms that encourage resource sharing, prevent envy, and ensure a fair distribution of resources. Incorporating these characteristics promotes collaboration, minimizes negative emotions, and prioritizes the well-being of all participants in the system.

This research introduces an innovative resource-allocation mechanism that addresses shortcomings in traditional methods. Our method prioritizes both fairness and efficiency in resource distribution, utilizing a token-based mechanism to ensure fairness and implementing individual preferences based on learned thresholds through an Actor-Critic method to improve efficiency. A computer simulation involving 40 accelerators and 20 agents in different environments demonstrates a performance improvement 1.28× compared to standard approaches. This study contributes by shedding light on the complex challenges of resource-allocation in heterogeneous systems and providing a practical solution with our approach.

Acknowledgements

I want to convey my appreciation to Dr. Seyed Majid Zahedi and Dr. Nachiket Kapre, my supervisors, whose guidance was instrumental throughout this project. The successful completion of this study would not have been feasible without their expertise.

A special acknowledgment goes to Professor Paul Ward and Professor Khuzaima Daudjee for dedicating their time to review my thesis.

I am deeply grateful to Aravind Vellora Vayalapra for his invaluable cooperation to the completion of this thesis; without his assistance, this work would not have come to fruition

I am deeply thankful to my friends, Amirabbas, Mohammad, MZi, Seyed Soheil, Soroosh, and Moridi, for their steadfast support and companionship during challenging moments in this journey.

Lastly, I would like to express my gratitude to my mother and brothers. Their unwavering love and support have been invaluable not only during this process but throughout my entire life.

Dedication

This thesis is dedicated to my absurd life, which has taught me that there is no point in any goal. Goals lose their value exactly when they are achieved, but the only thing that always remains is you. So, enjoy your existence in both sadness and happiness.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The performance of various machine learning applications is heavily dependent on the selection of the underlying hardware architecture [2, 8, 60, 62, 64]. Machine learning tasks, which encompass a wide range of applications, rely on the computational power [13, 27], scalability [38, 41, 46], memory, and storage capabilities [32, 43] of the hardware. Choices like using powerful GPUs or specialized hardware, optimizing for energy efficiency [27], and ensuring compatibility with machine learning frameworks can significantly impact the speed, efficiency, and overall effectiveness of the machine learning system [11, 42, 44].

Furthermore, applications often exhibit changing resource needs, sometimes demanding a significant amount of resources while at other times using them inefficiently. These variations in resource utilization can detrimentally affect the overall performance of a system [21, 37, 53, 59, 74, 77]. When a program is resource-intensive, it can compete with other programs for system resources, potentially causing delays and slowdowns. On the other hand, inefficient resource usage can lead to resource saturation and negatively impacting the user experience [33, 48]. Effectively managing and optimizing resource usage is crucial in system administration to maintain system stability and ensure a smooth user experience.

To address the variations in resource utilization among multiple applications, the practice of resource sharing is implemented [9, 53, 57, 66]. Resource sharing involves allowing different applications to collectively access and utilize hardware components and system resources

rather than operating in isolation. This collaborative approach can have several beneficial effects on system performance and resource allocation efficiency [49, 66]. Firstly, it promotes a more efficient utilization of system resources by enabling applications to share available resources as needed [53, 57]. Instead of each application trying to reserve and use a fixed set of resources, they can dynamically allocate and release resources, depending on their current needs [26, 53, 76]. This adaptability helps prevent resource shortages and over-provisioning, leading to better resource allocation efficiency.

Sharing resources among multiple applications presents a complex and multifaceted challenge at the intersection of performance optimization and equitable resource distribution [35, 36, 68]. Within a heterogeneous system environment characterized by diverse hardware components and software applications, the process of identifying and assigning tasks to suitable hardware resources can be inherently intricate [34, 74]. This intricacy is due to the multiple factors to consider, such as the computational capabilities of different hardware components, the memory requirements of applications, and the dynamic nature of workloads [77, 80]. As a result, the efficient utilization of these resources for task execution and overall performance enhancement can become a complex juggling act.

Moreover, the prevailing resource allocation strategies are often based on optimizing system utility, aiming to maximize overall performance and efficiency [23, 28, 45]. However, in this pursuit, fairness among users or applications can sometimes be overlooked [4]. Such resource allocation strategies may not ensure a level playing field for all users, potentially leading to a situation where some applications enjoy improved performance at the expense of others [9]. This lack of fairness can be particularly problematic in shared computing environments where multiple users or clients depend on shared resources. For example, in large companies, they get resources from cloud provider and let the employees to use for their jobs [53].

The inherent trade-off between performance optimization and fair resource allocation becomes a central concern [9, 36, 53]. On one hand, there is a pressing need to ensure that the system operates at peak efficiency, making the best use of available resources. On the other hand, the aspect of fairness cannot be dismissed, as it plays a critical role in preventing resource contention, ensuring user satisfaction, and adhering to principles of equity [23]. Striking the right balance between these competing objectives is a complex but

essential task in resource management.

Achieving fairness in a system necessitates the integration of specific characteristics within a mechanism of resource sharing, as identified by prior studies [23, 53, 82]:

- The mechanism should actively encourage agents to engage in resource sharing rather than relying solely on exclusivity [19, 71]. This means setting up rewards or special access for those who share. For instance, This reward can be a better performance in comparison with the exclusive usage.
- Fairness is upheld by ensuring that no agent experiences envy toward others [71]. This is so ideal to have this feature in our system and good performance together because everytime there is it is crucial that the mechanisms in place go beyond mere distribution of resources and actively work to prevent any user from feeling jealous or left out. This involves creating a system that allocates resources in a fair and transparent manner, ensuring that no individual or group is at a disadvantage or feels envious of others.
- A mechanism should propose a situation where no one can be better off without making someone else worse off. This means finding a balance where the distribution of resources is optimized in a way that benefits everyone involved. In such a system, any attempt to give more to one person or group would come at the expense of another. It is like ensuring that the sharing arrangement is as good as it can get for everyone, without favoring one user at the cost of another. This way, the system operates in a way that maximizes overall well-being and fairness.

By incorporating these characteristics into the system's design and operation, a foundation for fairness is established and promoting collaboration. However, achieving all three conditions simultaneously is not always feasible, especially when addressing one-shot problems. For instance, the second and third conditions may not be simultaneously achievable in all scenarios. Consider a scenario where we have two one-hour jobs in our system. Utilizing a first-in-first-out (FIFO) scheduler for these two jobs can satisfy the third condition. However, the second condition may not be met because the job running

second will envy the job running first. Conversely, employing a short time quantum and attempting to run these two jobs concurrently with a round-robin (RR) scheduler may not fulfill the third condition. In RR, both jobs will finish after two hours, whereas in FIFO, one job concludes after one hour, and the other after two hours, indicating that one job achieves better performance without detriment to the other. Therefore, we should satisfy most of these conditions as much as we can.

This research presents an innovative mechanism designed to overcome the shortcomings associated with conventional resource allocation methods. Unlike traditional approaches, our mechanism places a strong emphasis on both fairness and efficiency in resource distribution. To ensure fairness, our approach incorporates a token-based mechanism for allocating resources among agents. This mechanism introduces a systematic and equitable approach, enhancing the overall fairness in resource distribution within cloud accelerators.

Furthermore, to outperform existing mechanisms, we implement a strategy for distributing resources to agents based on their individual preferences. These preferences are derived from a threshold strategy, which is learned through the application of an Actor-Critic method [39]. An Actor-Critic method plays a pivotal role in shaping our resource preferences. By incorporating a learning mechanism, agents gain insights into optimal resource distribution, adapting their preferences based on real-time feedback.

We conducted a detailed examination of the effectiveness of our technique using a computer simulation. This simulation included 40 different accelerators through 20 agents in heterogeneous and homogeneous environments. For the accelerators, we use Gemmini [22] which is a full-system, full-stack DNN hardware exploration and evaluation platform Our assessment results are very encouraging, demonstrating a performance gain of $\sim 1.3\times$ in average when compared to standard approaches. In our test, in a situation that there is a big difference in sharing, our method gets $\sim 2.8\times$ better than the best schedulers available. Also, when the time limit is crucial, our method does $\sim 1.1\times$ better on average. When the time limit is not close, the performance becomes similar to the best solution available. These findings highlight the practical use and efficacy of our technique in the setting of heterogeneous accelerators.

We make substantial contributions in this study, beginning with a straightforward

description of the complex issue underlying the allocation of various resources. We dig into the difficulties that inevitably accompany managing these many types of resources, giving light on the intricacies that must be addressed for successful resource allocation in heterogeneous systems.

Our second contribution focuses on the development of fairness metrics tailored to heterogeneous systems. We not only develop these measures, but also demonstrate their suitability for the wide range of systems. By tailoring fairness metrics to the unique characteristics of heterogeneous setups, we aim to pave the way for a more equitable resource allocation process that considers the diverse needs of the system components.

In addition, we explain how our scheduler works and the technique we use to create thresholds dictating resource preferences among agents. This part of our contribution intends to improve resource allocation efficiency by proposing a systematic method to preference setting. To demonstrate the effectiveness of our suggested strategy, we report the results of performance and fairness assessments done in simulated environments in a systematic manner. These assessments provide a concrete indication of the practical applicability and beneficial impact of our contributions to diverse resource allocation.

Briefly, In Chapter 2, we explore our reasons for undertaking this project and why it is valuable. In Chapter 3, we clarify the concept of fairness in our project's framework and attempt to define it for our purposes. Chapter 4 delves into our algorithm and its functioning. In Chapter 5, we present the results of our system's evaluation. Lastly, in Chapter 6, we discuss existing works related to ours and how we address their limitations.

Chapter 2

Motivation and Background

This chapter aims to discuss the motivation behind our work. We begin by exploring hardware accelerators and their varying performance levels when executing ML applications due to their heterogeneous nature. Subsequently, we dive into the concepts of fairness and efficiency and examine their interconnectedness, particularly within heterogeneous systems. We highlight the challenges in achieving a balance between fairness and optimizing system performance, emphasizing the complexities involved in satisfying both objectives simultaneously within heterogeneous computing environments.

2.1 Heterogeneous Hardware Accelerators

Machine learning applications are becoming increasingly popular, notably in fields such as computer vision, speech recognition, and robotics [26, 53, 76, 77, 80]. As the complexity of machine learning models grows, the need for fast hardware accelerators to meet their rising computing requirements becomes critical [47, 52, 61, 79]. Using customized hardware accelerators is one of the commonly accepted options. Various companies provide different kinds of hardware accelerators appropriate for machine learning activities. These accelerators extend beyond GPUs to include a variety of specialized devices designed to improve the performance of machine learning tasks.

These accelerators are priced differently, with options ranging from low-cost to high-end [15, 55]. Typically, the cost is proportional to the increased capabilities of the accelerators. However, the costly prices of modern accelerators make it difficult for many small businesses and research organizations to invest in such high-performance technology. Cloud service providers have stepped in to give rental choices for these pricey accelerators in order to address this issue [55]. This enables smaller entities, such as research teams and budding firms, to have access to and use sophisticated gear without making a large initial expenditure. Consequently, small companies and research groups can now leverage cloud-based solutions to rent and run their applications on these specialized hardware accelerators, democratizing access to high-performance computing resources in the field of machine learning.

2.1.1 Gemmini

Gemmini [22] is a full-stack DNN accelerator generator that is open-source and meant to allow rigorous assessment of deep learning architectures. It solves the problem of assessing DNN accelerators by offering a versatile hardware template, a multi-layered software stack, and an integrated SoC environment.

Architecture

Figure 2.1 depicts Gemmini's architectural plan, which is centred on a spatial design with processing elements (PEs) scattered throughout. These PEs execute dot products and accumulations. Data is retrieved from a local scratchpad of banked SRAMs and stored in a local accumulator with a bitwidth bigger than the inputs. Gemmini's adjustable peripheral circuitry supports various DNN kernels such as pooling [24] and non-linear activations. These accelerators may be programmed and configured thanks to their integration with a RISC-V host CPU.

As shown in Figure 2.2, Gemmini's spatial array is constructed with a dual-level hierarchy, providing flexibility for alternative microarchitecture designs [22]. The first level is made up of tiles connected by explicit pipeline registers, and inside these tiles are arrays of PEs linked in combination. Using weight- or output-stationary dataflow, each PE performs a

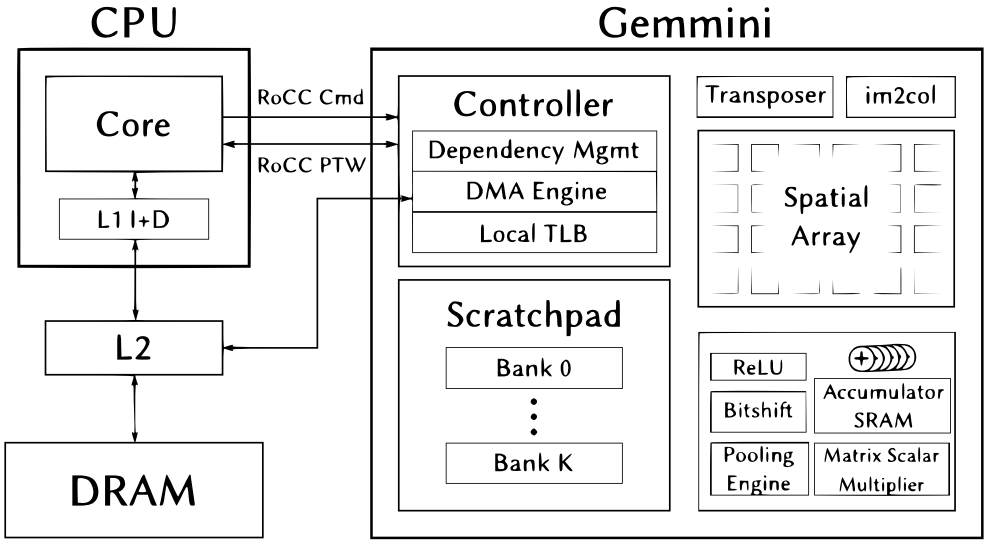


Figure 2.1: Gemmini hardware architectural template overview [22]

multiply-accumulate operation once every cycle. Tiles are connected combinationally in rectangular arrays of PEs, whereas the spatial array forms a rectangle array of these tiles with pipeline registers between them. Only nearby PEs and tiles share inputs and outputs.

Programming Support

Gemmini's generator generates both hardware and a bespoke software stack, increasing developer efficiency while dealing with multiple hardware shapes. It provides a multi-level software method that includes a high-level, user-friendly flow that turns ONNX [14] file DNN descriptions into executable software while optimizing kernel mapping on the Gemmini accelerator. It also has a low-level alternative with C/C++ APIs and optimized functions for common DNN kernels. To achieve maximal performance, these routines require particular tuning for each hardware instance, taking into account characteristics such as scratchpad sizes. Each new accelerator iteration includes a header file that describes parameters such as spatial array size, allowed dataflows, and computational blocks.

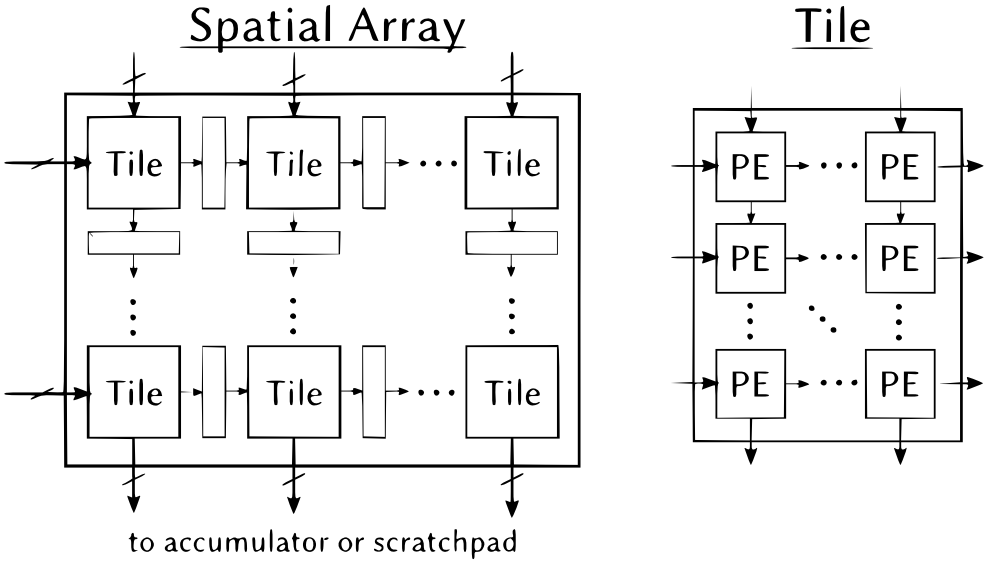


Figure 2.2: Microarchitecture of Gemmini's two-level spatial array [22]

System Support

Using the Chipyard [3] framework, Gemmini enables the coupling of RISC-V CPUs with Gemmini-developed accelerators. This integration varies from simple microcontrollers that handle mostly IO duties to complex, high-performance CPUs that manage numerous demanding applications while running the Gemmini accelerator. Multiple CPUs and accelerators in a system can operate on various tasks at the same time. Bus widths, cache sizes, and memory system architectures might also differ amongst system configurations. Integrating Gemmini accelerators with a full RISC-V SoC and deep software stack, including an operating system, allows for more realistic testing of workloads [22]. This setup reveals potential issues that wouldn't appear in simpler setups.

2.2 Fairness vs. Efficiency

Balancing performance and maintaining fair resource allocations across time periods is an ongoing and challenging task [35, 36, 68]. This problem arises mostly from the performance

reduction that frequently occurs when attempting to allocate resources equally, especially when there is inadequate or no knowledge about the value of diverse workloads [9]. When the goal is to maximize efficiency, it often leads to a situation in which a few applications dominate the bulk of the resources. While this strategy is efficient, it may result in an unbalanced and potentially unfair distribution in which some applications are given preference over others [72, 73].

Prioritizing fairness in resource distribution, on the other hand, might create its own set of issues [36, 53]. While this method guarantees that resources are spread more equitably among all programs, it may not necessarily result in the best use of those resources. In such circumstances, the system's overall output or performance may be lower than what might be achieved if resources were distributed based on efficiency or workload needs. This trade-off between fairness and efficiency is an important concern in system design, particularly when resources are limited and the interests of different applications or users must be balanced. The aim is to establish a happy medium that maximizes overall system performance while preserving a sufficient amount of resource fairness.

2.2.1 Early Scheduler

Traditionally, early scheduling algorithms, such as the round-robin algorithm, did not place a great value on efficiency. Instead, these systems cycled through a list of jobs or users, allocating an equal amount of resources or processing time to each before moving on to the next. While simple and fair in its rotation, this strategy frequently neglected the diverse demands and potential efficiency of different activities [58, 65]. It did not account for the fact that some jobs may take more resources or be finished more rapidly, resulting in inefficient resource use and possible bottlenecks in systems with considerably varying task requirements.

2.2.2 Recent Scheduler

More modern systems have begun to incorporate performance factors with fairness. Dominant Resource Fairness (DRF) [23], for example, was designed to allocate resources in a

cluster while ensuring fairness based on the dominant resource required by each job. DRF provides a step forward in thinking by seeking to balance the demands of various jobs with the available resources, with a focus on the most crucial resource for each work. Despite its gains, DRF still prioritizes fairness above absolute efficiency [4]. Another approach, as proposed in [81], is an algorithm designed for sharing heterogeneous resources among multiple agents. This strategy focuses on allocating strong cores to agents who actively bid for them, implying a market-like process for resource allocation. While innovative, this approach is mainly tailored for systems that combine powerful and typical resources. Its applicability is limited in scenarios involving more than two types of resources, thus restricting its usefulness in more complex or diverse environments.

2.2.3 Issue of Scheduling in Heterogeneity

Scheduling resources in a system that includes heterogeneous resources is challenging primarily due to the diverse characteristics of the resources involved [10, 34]. In a heterogeneous environment, resources differ significantly in their capabilities and performance metrics [40]. For instance, some resources might have higher processing power, while others might offer better memory or specialized hardware for specific tasks like graphics processing or data encryption. This diversity means that each resource is uniquely suited for certain types of tasks but might be less efficient or even incapable of handling others. As a result, the scheduler must have a detailed understanding of both the requirements of the tasks and the capabilities of each resource.

Moreover, balancing load across these heterogeneous resources adds another layer of complexity [72]. The goal of load balancing is to maximize resource utilization and minimize overall system latency by distributing tasks in a way that avoids overloading any single resource while underutilizing others [31]. This requires a sophisticated scheduling algorithm that can not only match tasks to the most appropriate resources but also continuously monitor and adjust the distribution of tasks in response to changing conditions. Such an algorithm needs to consider the current load on each resource, the predicted future load, and the potential impact of each task on resource performance. In a heterogeneous environment, the varying performance characteristics of each resource make this a particularly challenging

Table 2.1: Arrivals of queues

ROUNDS	Arr_1	Arr_2
r_1	150	275
r_2	370	280

task [70]. The scheduler must effectively balance the load while ensuring that each task is executed on a resource that can handle it efficiently.

Example

In this scenario, two agents are involved in a system where they need to manage the utilization of two hardware accelerators. Each agent oversees a queue containing multiple applications, all of which require processing on these accelerators. For clarity, we denote each resource (hardware accelerator) as R_i, where i represents the specific resource in question. The complexity of this setup lies in the fluctuating workload of each agent's queue, which varies at different times. To simplify our discussion, we consider only two distinct applications in this example. These applications represent the variety of tasks that need to be processed by the accelerators. To better understand the dynamics of this system, we refer to three key tables. The first, Table 2.1, provides detailed insights into the arrival of each agent's queue. These categorically list the number of tasks present in the queues of each agent at any given time, offering a snapshot of the workload distribution between the two agents. Meanwhile, Table 2.2 delves into the performance aspect, specifically focusing on the throughput each resource delivers for the different applications. This table is crucial for understanding how effectively each hardware accelerator handles the various applications, which in turn influences the agents' strategies for queue management and resource allocation. By analyzing both tables, one can gain a comprehensive view of the operational efficiency and task distribution in this two-agent, two-accelerator system.

In this system, the primary goal for each agent is to minimize the total load of all applications. This objective is rooted in the belief that a lower aggregate load reduces the

Table 2.2: Agents' throughput for each resource

Agents	R_1	R_2	R_3	R_4
A_1	100	120	150	200
A_2	200	100	50	200

likelihood of system failures, thereby enhancing overall efficiency and reliability. Consequently, the focus is not just on managing individual tasks but on reducing the combined burden of all tasks across the system.

To achieve this, we consider a scenario that emphasizes fairness in resource allocation. In an ideal fair scheduling system, the time allocated to each agent for using each resource would be evenly divided. This equitable approach ensures that each agent receives an equal opportunity to utilize the resources, thereby maintaining balance in the system. In practice, this means that in each scheduling round, each agent would be allocated exactly half of the total available time on each resource. Such an arrangement implies that the throughput an agent can achieve for each application is effectively halved during their allocated time. The rationale behind this is to ensure that no single agent monopolizes the resources, allowing for a fair and balanced distribution of processing time. This assumption of equal time allocation forms the basis of our scheduling strategy, focusing on fairness and balanced resource usage. It is a strategy that not only ensures equal access to resources but also aligns with the agents' preference for minimizing overall load to reduce the risk of system failure.

Referencing Table 1, we observe the workload distribution for two agents over the initial two rounds of job processing. In Round 1, Agent 1 receives 150 jobs in its queue, while Agent 2 experiences a higher influx with 275 jobs. This disparity in job intake significantly impacts the processing dynamics under the fair scheduler system. Under the mentioned scheduling system, which equally divides resource time between the two agents, Agent 1 effectively manages to clear its entire queue of 150 jobs in Round 1, leaving its queue empty. In contrast, Agent 2, grappling with a larger workload, is only able to process 225 out of its 275 jobs. This results in a backlog of 50 jobs that are carried over to the next round.

Moving to Round 2, the scenario evolves. Agent 1 now faces a substantial increase in its workload, with 370 new jobs entering its queue. Given the constraints of the scheduling system, it can only process 285 of these jobs, resulting in 85 jobs remaining unprocessed by the end of the round. Meanwhile, Agent 2 starts Round 2 with a combined total of 330 jobs, consisting of 280 new jobs and the 50 unresolved jobs from Round 1. However, similar to the previous round, Agent 2 is only capable of processing 225 jobs, leaving a higher residue of 105 jobs for the subsequent round.

Assessing the performance in terms of utility, a measure of the agents' effectiveness in clearing their queues, we notice a diminishing trend. In Round 1, the utilities for Agents 1 and 2 are 0 and -50, respectively, reflecting Agent 1's ability to clear its queue and Agent 2's backlog. In Round 2, the utilities further decline to -85 for Agent 1 and -105 for Agent 2. This downturn indicates an increasing inefficiency in handling the incoming workload under the current fair scheduling system, as both agents accumulate a growing number of unprocessed jobs.

In this alternative scenario, we explore a scheduling strategy that deviates from strict fairness but seeks to optimize overall efficiency for both agents. This approach involves allocating resources in a manner that better matches each agent's workload and throughput capabilities. In the first round, the resource allocation is adjusted to better suit the agents' incoming job loads. Specifically, Agent 1 is allocated resource R_3, while Agent 2 is given resources R_1, R_2, and R_4. This tailored allocation allows both Agent 1 and Agent 2 to successfully process all their jobs in the first round. As a result, both agents achieve a utility of 0, indicating no backlog and complete processing of their respective queues. In the second round, the resource allocation is again strategically adjusted. Agent 1 is given resources R_3 and R_4, while Agent 2 is allocated R_1 and R_2. In this configuration, Agent 1 is able to process a significant portion of its jobs, totaling 350 out of 370. However, it still faces a minor shortfall, leaving 20 jobs unprocessed. On the other hand, Agent 2, with its allocated resources, manages to clear all of its jobs, maintaining a utility of 0.

Comparing this scenario with the previous one, it is evident that both agents benefit more from the revised scheduling strategy. In the first scenario, under strictly fair scheduling, both agents experienced a backlog, indicated by negative utility values. In contrast, in this new scenario, the tailored allocation of resources leads to improved utility scores for both

agents. Agent 1 has a significantly reduced backlog, and Agent 2 is able to consistently clear its queue. This comparison highlights the effectiveness of a more flexible scheduling approach that prioritizes efficiency and job throughput over strict equality in resource distribution.

Chapter 3

Fairness

Drawing on the core notions of economic game theory as elaborated in many past academic works, it becomes important to achieve three separate requirements in order to construct an equitable and sustainable framework of fairness [82]. These factors are critical in ensuring that the outcomes of any economic game are regarded as equitable and acceptable by all parties involved, encouraging cooperation and mutual respect.

The first of these three principles is known as Sharing Incentives (SI) [19, 71]. This concept proposes a model in which all participants in the economic game favour a community approach to resource allocation. In essence, it implies that people would get more value and advantage from collectively sharing their resources rather than using them in a solo, exclusive manner. This approach promotes a culture of common benefit and discourages resource hoarding or selfish use, ensuring that benefits are distributed more fairly among all players.

The following rule is about being Envy-Free (EF), which is important to the idea of fairness in economic game theory. When resources or benefits are spread among the participants, no one player has a desire for what has been allotted to the others, according to this rule [71]. In a more detailed sense, this rule assures that no participant believes that another has obtained a better bargain or a larger share of resources. The purpose of this approach is to inspire in all participants a sense of happiness and satisfaction with

what they have received. It's a critical component in maintaining harmony and preventing feelings of resentment or unfairness in the distribution process.

Pareto Efficiency (PE) is the third and equally important principle [23, 82]. This notion is a little more complicated, requiring a fine balance in the distribution of resources or happiness. Pareto Efficiency is accomplished when resources are allocated in such a way that it is impossible to make one person happier or better off without concurrently making at least one other person unhappy or worse off. This approach provides an optimal resource allocation in which the collective welfare is maximized, and no more improvements can be made without harming at least one member.

We initiate an in-depth examination of each principle in detail. In section 3.2, we delve into the concept of the sharing incentive and its formulation. Moving on to section 3.3, we explore the envy-free principle. Finally, in section 3.4, we discuss the role of Pareto efficiency in our system.

3.1 Preliminaries

Our research focuses on the analysis of a system that includes m distinct machine learning accelerators. The goal is to allocate these accelerators among n users across a sequence of R rounds. In each of these rounds, a user denoted by i may gain access to several accelerators. Each accelerator provides different levels of utility, depending on the specific workload of the user. This leads to the formation of preference sets P_i^r for the accelerators by each user. We use the symbol u_i to denote the total utility that an agent, or user i, gains over all R rounds. To break it down, u_i^r represents the utility that agent i acquires in a single round r. Furthermore, $u_i^{r,c}$ indicates the potential utility that agent i could receive from accelerator c during round r. In each round, we represent the queue length of agent i by q_i^r and its arrival by Arr_i^r in round r. Therefore, u_i is the sum of all u_i^r values, and each u_i^r is the combined sum of $u_i^{r,c}$ values assigned to agent i:

$$u_i = \sum_{r \in R} u_i^r \tag{3.1}$$

$$u_i^r = -max(0, q_i^r + Arr_i^r - \sum_{c \in M} x_i^{r,c} \cdot u_i^{r,c}) \tag{3.2}$$

Where M is the set of all accelerators and $x_i^{r,c} \in \{0, 1\}$ denotes if accelerator c is allocated to agent i in round r or not.

3.2 Sharing Incentive

In a situation where multiple individuals or groups (referred to as agents) use a common set of resources, our designed system aims to promote a culture of cooperation and mutual benefit [50, 82]. The goal is to encourage all agents to prefer working together and sharing resources instead of using them separately and exclusively [18, 50, 83]. The success of our system is measured by what we call the Sharing Incentive (SI) [71]. SI is achieved when all agents involved are happy with the way resources are being shared. This means every agent feels that they are getting enough from the shared resources and sees the benefit in continuing to share.

On the other hand, if even one agent feels that they are not getting their fair share or are unhappy with the way resources are allocated in our system, it indicates that we have not successfully met the Sharing Incentive. In such cases, it shows that our system needs improvement to ensure everyone feels content with the shared use of resources. Our ultimate aim is to create a balance where all agents see the value in sharing resources, leading to a more efficient and harmonious use of what is available.

To formalize SI, we introduce two distinct types of utilities for each agent: shared utility and expected utility.

- **Shared Utility** (u_i^{sh}): This represents the utility or benefit that agent i receives when participating in a resource-sharing arrangement with other agents.It reflects the value or satisfaction derived from a collaborative use of resources.

- **Expected Utility** (u_i^{ex}): In contrast, this denotes the utility that agent i would gain if it had exclusive access to all resources with regard to its portion in sharing.It provides

Table 3.1: An example of agents' resource need and utilities

Agents	Preference (P_i^r)	Utilities ($u_i^{r,c}$)
A_1	[2,4,3,1][1,2,3,4][3,1,2,4]	[0.1,0.9,0.1,0.2][0.9,0.9,0.9,0.6][0.8,0.3,0.9,0.1]
A_2	[4,2,3,1][4,2,3,1][1,2,4,3]	[0.5,0.6,0.5,0.7][0.4,0.5,0.6,0.6][0.6,0.5,0.1,0.5]

a baseline to compare the benefits of shared versus expected resource usage.For example if two agents want to share a resource and they have equal portion for usage, their u^{ex}s is the utilities they gain when they just have half time usage of resource in each round.

To assess how fair and effective our resource allocation mechanism is, we introduce a metric known as *utility-fairness*. This metric is defined for each agent i as the ratio of shared utility to expected utility, mathematically represented as:

$$\phi_i = \frac{u_i^{sh}}{u_i^{ex}} \tag{3.3}$$

The utility-fairness metric, ϕ_i, helps in understanding how the benefits of shared resource usage compare to the benefits of expected use for each agent.

A mechanism is said to satisfy the Sharing Incentive (SI) if it meets the following condition: The mechanism satisfies SI if for every agent i, the shared utility is at least as great as the expected utility. Mathematically, this can be represented as:

$$\phi_i \geq 1, \quad \forall i \tag{3.4}$$

To be more specific, in the absence of SI, agents desire equal weighted resource division. This allocation results in inefficient resource use. This formula is applicable when the utility function represents agents' gains, such as throughput. If the utility function instead represents the costs an agent must bear, like latency, then the equation should be expressed as $\phi_i \leq 1$.

Table 3.2: Sequences of allocations in 3 rounds for each agent and their overall utilities

Name	A_1 Allocation (x_1)	A_2 Allocation (x_2)
Time Equal	$[\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}][\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}][\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}]$	$[\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}][\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}][\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}]$
Dynamic-Fair Part. (M_0)	[0,1,0,1][1,1,0,0][1,0,1,0]	[1,0,1,0][0,0,1,1][0,1,0,1]
Fixed-Fair Part. (M_1)	[0,0,1,1][0,0,1,1][0,0,1,1]	[1,1,0,0][1,1,0,0][1,1,0,0]
MTF (M_2)	[0,1,0,0][1,1,1,0][1,0,1,0]	[1,0,1,1][0,0,0,1][0,1,0,1]

Table 3.3: Overall utilities

Name	u_1	u_2
Time Equal	3.35	3.05
Dynamic-Fair Part. (M_0)	4.6	3.2
Fixed-Fair Part. (M_1)	2.8	3.1
Most Token First (MTF) (M_2)	5.3	3.3

To further illustrate the concept of *utility-fairness*, consider an example involving the resource preferences of two agents. These preferences are tabulated in Table 3.1. As a specific instance, in the first round of allocation, agent two demonstrates a preference order where accelerator c_4 is favored over c_2, which in turn is preferred over c_3, and so forth. (In this example, we assume that we have a maximum number for the length of queue and normalize the length by this maximum number. We have more detail about this in section 5)

Additionally, we illustrate various allocation scenarios that may arise under different mechanisms in Table 3.2. Let us focus on a scenario involving two agents, denoted as A_1 and A_2. For the sake of simplicity in this example, we assume that both agents have equal weight/portion in the decision-making process, with $w_1 = 0.5$ and $w_2 = 0.5$ respectively. Their primary objective is to allocate the usage of a single Graphics Processing Unit (GPU) between them in a manner that is perceived as fair by both parties.

At the outset, let us consider the expected utilities of the agents, denoted as u_1^{ex} and

u_2^{ex}. These utilities represent the satisfaction or benefit each agent would derive if they were to use the GPU with expecting of allocation based on their weight. Initially, we can say $u_1^{ex} = 3.35$ and $u_2^{ex} = 3.05$.

However, the scenario changes when we introduce a dynamic-equal partitioning mechanism, denoted as M_0. In this setup, the GPU is divided equally between the two agents. Under such a partitioning scheme, the shared utilities of the agents are denoted as $u_1^{sh} = 4.6$ and $u_2^{sh} = 3.2$.

The utility-fairness metric, ϕ_i, for each agent is calculated as the ratio of their shared utility to their expected utility. For the first agent, $\phi_1 = \frac{u_1^{sh}}{u_1^{ex}} = 1.36$, and for the second agent, $\phi_2 = \frac{u_2^{sh}}{u_2^{ex}} = 1.04$. These values suggest that both agents receive more utility than the utility they would have if they used the GPU by time-equal allocation.

Continuing our exploration of resource allocation mechanisms, we now consider mechanism M_1, which is characterized as a fixed-equally partitioning mechanism. Under M_1, the shared utilities for agents A_1 and A_2 are denoted as $u_1^{sh} = 2.8$ and $u_2^{sh} = 3.1$, respectively. This reflects the utility each agent receives when the accelerator is partitioned equally but in a fixed manner, without dynamic adjustments.

The utility-fairness metric for each agent under mechanism M_1 can be computed as follows:

- For agent A_1: $\phi_1 = \frac{u_1^{sh}}{u_1^{ex}} = 0.82$.
- For agent A_2: $\phi_2 = \frac{u_2^{sh}}{u_2^{ex}} = 1.02$.

It is evident from these metrics that agent A_2 exhibits a preference for mechanism M_1 over M_0, the dynamic-equal partitioning mechanism. On the other hand, agent A_1 shows a preference for either the equal partitioning scenario (M_0) or the time-equal use scenario. This divergence in preferences among the agents underlines that mechanism M_1 fails to satisfy the Sharing Incentive (SI), as it does not equally cater to the interests of both agents.

Next, we introduce our proposed allocation mechanism, *Most Token First* allocation, denoted as M_2. We go through the allocation mechanism in section 4. Under M_2, the

shared utilities are $u_1^{sh} = 5.3$ for agent A_1 and $u_2^{sh} = 3.3$ for agent A_2. This leads to the following utility-fairness metrics:

- for agent A_1: $\phi_1 = \frac{u_1^{sh}}{u_1^{ex}} = 1.58$.
- for agent A_2: $\phi_2 = \frac{u_2^{sh}}{u_2^{ex}} = 1.08$.

These values indicate that, similar to mechanism M_0, both agents receive more utilities than the utilities they would have achieved under time-equal use. This alignment in agent preferences suggests that mechanism M_2 successfully satisfies the requirement of the Sharing Incentive, making it a viable option for fair resource allocation between the two agents.

3.3 Envy Free

Envy-freeness (EF) is a concept often used in the field of economics and game theory, which pertains to the allocation of resources or goods among multiple agents. In an envy-free scenario, each agent is allocated a portion of the resources, and the core of this idea revolves around the satisfaction and preference of each agent towards their allocated share compared to the shares of others [16, 29, 71]. An agent experiences envy if they desire the allocation assigned to another agent, often because they perceive it to be better or more beneficial for their interests.

The dynamics of EF become particularly interesting and complex when the allocations are non-uniform, and the perceived value of these allocations varies from one agent to another. This difference in valuation can be attributed to a variety of factors, such as individual preferences, needs, or the utility each agent derives from their allocation. For instance, in a situation where resources are scarce, an agent might prefer another's allocation because it better satisfies their specific needs or because it offers a higher utility.

EF is closely tied to the concept of fairness in allocation [30]. A fair allocation is often described as one where no agent would prefer the allocation of another, implying that everyone is at least as happy with their allocation as they would be with the allocation of

another agent. This does not necessarily mean that all allocations are equal, but rather that they are equitable in the eyes of the agents involved.

When evaluating whether an allocation satisfies the condition of being Envy-Free (EF), it is crucial to consider the subjective perceptions of the agents. If all agents exhibit a preference for their individual allocations, it signifies that the allocation satisfies the EF condition. This is a strong indication of a successful and fair distribution of resources, as it implies that each agent believes they have received a fair share, and there is no desire to exchange their allocation for that of another agent.

To formalize the concept of EF within our system, let us consider a set of agents denoted by $A = \{A_1, A_2, ..., A_n\}$ and a set of resources or accelerators that can be allocated. Let $X = \{X_1, X_2, ..., X_n\}$ represent the allocation of resources to these agents over a series of R rounds using a specific mechanism M. In this context, X_i refers to the allocation for agent A_i, and is defined as $X_i = \{X_i^1, X_i^2, ..., X_i^R\}$, where X_i^r represents the allocation to agent A_i in round r. Each X_i^r is a set $\{x_i^{r,1}, x_i^{r,2}, ..., x_i^{r,m}\}$, where $x_i^{r,c}$ is a binary indicator taking a value in $\{0, 1\}$, denoting whether accelerator c is allocated to agent i in round r.

Furthermore, we define a utility function $u_i(X)$ to quantify the satisfaction or benefit that agent A_i derives from a certain allocation X. Specifically, $u_i^{X_j}$ is the utility experienced by agent A_i when utilizing the system with the allocation originally designated for agent A_j.

The mechanism M is said to satisfy EF if and only if the following condition holds for all pairs of agents:

$$\forall i, j \in \{1, 2, ..., n\}, \quad w_j \cdot u_i^{X_i} \geq w_i \cdot u_i^{X_j} \tag{3.5}$$

In this equation, w_i represents the weight assigned to agent A_i. The condition essentially states that for each agent A_i, the utility they derive from their own allocation X_i, when weighted by their importance w_i, should be at least as great as the utility they would derive from any other agent's allocation X_j. This ensures that no agent prefers the allocation of another agent over their own, thus satisfying the criteria of EF in the allocation.

Consider the aforementioned scenario involving two agents, A_1 and A_2, in the context of a resource allocation problem. Under mechanism M_1, an alteration in the allocations of

A_1 and A_2 leads to the following utility outcomes: $u_1^{X_2} = 3.9$ and $u_2^{X_1} = 3.0$. Notably, in this mechanism, agent A_1 achieves a higher utility when allocated the resources of A_2 (3.9) compared to their utility from their own allocation (assumed to be lower). This discrepancy indicates that A_1 envies the allocation of A_2, as they perceive it to be more beneficial or satisfying.

Alternatively, in mechanism M_2, upon altering the allocations, the resulting utilities are $u_1^{X_2} = 1.4$ and $u_2^{X_1} = 2.8$. In this scenario, both agents demonstrate a higher preference for their own respective allocations over those of the other agent. Specifically, the utility A_1 derives from A_2's allocation is less than the utility from their own allocation, and similarly, A_2 finds their own allocation more satisfying than that of A_1. This outcome implies that neither agent envies the other's allocation, thus fulfilling the condition for EF. Consequently, while mechanism M_1 fails to meet the EF criterion due to the presence of envy from A_1 towards A_2, mechanism M_2 successfully aligns with the EF criterion.

3.4 Pareto Efficiency

Pareto Efficiency (PE) is a fundamental concept in economics and resource allocation that plays a crucial role, especially in heterogeneous environments where resources vary in type and utility [67]. Within any given allocation mechanism, PE embodies the idea that it is impossible to improve the utility (or satisfaction) of a single agent without simultaneously reducing the utility of at least one other agent [7, 67]. This concept is pivotal in understanding the dynamics of resource allocation, particularly in systems where resources are limited and diverse in nature [54].

PE revolves around the goal of achieving an optimal allocation among all feasible distributions of resources [12]. An allocation is said to be Pareto optimal if there is no other feasible allocation that could make at least one agent better off without making someone else worse off. In other words, a Pareto optimal allocation is a state where resources are distributed in such a way that no further reallocation can enhance the utility of one agent without harming another. This principle ensures that resources are utilized in the most efficient manner possible within the given constraints.

To formalize the concept of Pareto Efficiency (PE) within the framework of our resource allocation system, we introduce specific notations and conditions that define PE in the context of heterogeneous resource distribution across different rounds. Let $\Psi = \{X^1, X^2, ..., X^k\}$ represent the set of all possible resource allocations among the agents, where k is the total number of feasible allocations. Each allocation X^i in Ψ is a distinct distribution of resources among the agents.

Furthermore, for each allocation X^i and round r, we define a utility set $U^{X^i,r} = \{u_1^{X^i,r}, u_2^{X^i,r}, ..., u_n^{X^i,r}\}$, where $u_j^{X^i,r}$ denotes the utility of agent j in round r when allocation X^i is implemented. The utility $u_j^{X^i,r}$ quantifies the satisfaction or benefit derived by agent j from the specific allocation X^i during that round.

Considering a specific mechanism $\hat{M}$ employed in our system, let $X^{\hat{M}}$ be the allocation chosen by $\hat{M}$ for a particular round r. We say that the mechanism $\hat{M}$ satisfies Pareto Efficiency in round r if and only if the following condition holds:

$$\nexists M, M \neq \hat{M} \Rightarrow (\forall i \quad u_i^{X^M,r} \geq u_i^{X^{\hat{M}},r}) \wedge (\exists j \quad u_j^{X^M,r} > u_j^{X^{\hat{M}},r}) \tag{3.6}$$

This condition states that there does not exist any other allocation X^i in the set of possible allocations Ψ that can make every agent strictly better off compared to the allocation $X^{\hat{M}}$ chosen by the mechanism $\hat{M}$. In other words, $X^{\hat{M}}$ is PE if improving the utility of any one agent under any alternative allocation X^i would necessarily reduce the utility of at least one other agent.

To illustrate the concept of Pareto Efficiency (PE) using a concrete example, let us consider two mechanisms, M_1 and M_2, in the context of our allocation system. We analyze the utilities derived by two agents in the first round of allocation under these mechanisms to assess whether they satisfy the criteria of PE.

Under mechanism M_1, the utilities of the two agents in the first round are given by: $u_1^{M_1,1} = 0.3$ and $u_2^{M_1,1} = 1.1$. Here, $u_j^{M_1,1}$ denotes the utility of agent j in the first round under allocation mechanism M_1. According to the principle of PE, if there exists an allocation M such that $u_1^{M,1} > 0.3$ while maintaining $u_2^{M,1} \geq 1.1$ or vice versa, it indicates a failure of M_1 to satisfy PE. This failure arises because the existence of such an alternative

allocation implies that it is possible to improve the overall welfare of all agents without disadvantaging any individual agent, a fundamental requirement for an allocation to be Pareto efficient. In this example, mechanism M_2 presents such an alternative. Under M_2, the utilities for the first round are given by: $u_1^{M_2,1} = 0.9$ and $u_2^{M_2,1} = 1.7$.

Comparing the utilities under M_2 with those under M_1, we observe that both agents achieve higher utilities under M_2 than under M_1 in first round.($0.9 > 0.3$ for agent 1 and $1.7 > 1.1$ for agent 2). This indicates that M_2 is a more efficient allocation in terms of maximizing the welfare of all agents without making any one of them worse off, hence satisfying the criteria of Pareto Efficiency for the first round of allocation.

Chapter 4

Mechanism Design

We provide a token-based system that collects agents' preferences and allocates resources based on their token amounts. Agents also learn how to report their preferences in order to maximize their own usefulness. In this part, we will look at our system's scheduler and learning process.

4.1 Most-Token-First Scheduler

4.1.1 Tokens Distribution

Our scheduler is designed to operate in a dynamic environment, where resource allocations are determined on a round-by-round basis. This approach allows for adaptability and responsiveness to changing conditions and requirements of agents. In each round, the scheduler takes into account the preferences of the agents and the availability of tokens. The agents are equipped with individual budgets, denoted as B_i^r, for each round r. These budgets are crucial for determining how resources are allocated among the agents. The initial budget allocation (B_i^1) for each agent is carefully determined in proportion to their respective weights. Let us assume that the total quantity of tokens available in the system is represented by T. The weight of an agent i is denoted as w_i, and it plays a significant

role in the distribution of the initial budgets. The formula for calculating the initial budget of agent i is given by:

$$B_i^1 = \frac{w_i \cdot T}{\sum_{j=1}^{n} w_j} \tag{4.1}$$

In this equation, the denominator $\sum_{j=1}^{n} w_j$ represents the sum of weights of all agents in the system. It ensures that the distribution of the initial budgets is relative to the total weights.

Furthermore, agents are allowed to bid for resources using their allocated budgets. The bidding process is governed by rules that ensure fair competition and efficient utilization of resources. The scheduler evaluates these bids and allocates resources accordingly, It not only aims to maximize the overall utility and satisfaction of all agents, but also adheres to the constraints of token availability and budget limitations.

4.1.2 Preferences and Thresholds

In the context of our resource allocation system, the process of agents expressing their preferences plays a pivotal role in each round of allocation. This is achieved by agents submitting an ordered list of preferences for the available m accelerators. For each agent i in round r, this set of preferences is denoted by $P_i^r = \{p_i^{1,r}, p_i^{2,r}, ..., p_i^{m,r}\}$. The list P_i^r consists of m elements, each representing a accelerator in the set C. These elements are ordered to reflect the preference hierarchy of the agent, such that for any two accelerators $p_i^{k,r}$ and $p_i^{l,r}$ in C, the accelerator $p_i^{k,r}$ is preferred over $p_i^{l,r}$ whenever $k < l$. This ordering is crucial as it guides the scheduler in understanding the priorities of each agent regarding the accelerators.

In addition to providing their preference list, each agent specifies a threshold value τ_i. This threshold value is a critical parameter for the scheduler as it represents the minimum level of utility, denoted as $u_i^{r,c}$, that an agent is willing to accept for allocating their tokens to a particular accelerator. In essence, τ_i acts as a cut-off point, indicating the agent's willingness to commit resources to their preferences. If the utility $u_i^{r,c}$ for a accelerator c

is equal to or exceeds this threshold value τ_i, the agent is amenable to allocate tokens to secure that accelerator. Conversely, if the utility for a accelerator is below this threshold, it implies that the agent recognize the accelerator as less desirable or not worth.

The scheduler plays a crucial role in interpreting these preferences and threshold values. It utilizes this information to make informed decisions about resource allocation. Specifically, the scheduler will not charge an agent for a accelerator if the utility value $u_i^{r,c}$ associated with that accelerator falls below the agent's threshold τ_i.

4.1.3 Algorithm

In our system, each round of resource allocation is characterized by a structured process where agents actively participate by submitting their preferences and corresponding threshold values. This process is integral to the functioning of our allocation mechanism, as outlined in algorithm 1. The mechanism operates in a sequential manner, taking into account both the preferences of the agents and their available budgets.

The first step in this process involves the mechanism selecting an agent who not only has the highest budget available but also expresses a desire for at least one accelerator where the utility $u_i^{r,c}$ is greater than or equal to their specified threshold τ_i. This step ensures that the agent with the most resources at their disposal and a clear preference for high-utility accelerators is given priority in the allocation process.

Once such an agent is identified, the mechanism proceeds to allocate to them their most preferred accelerator. This is determined based on the agent's submitted preference list P_i^r, where the topmost accelerator that meets the utility threshold τ_i is selected. Following this allocation, the allocated accelerator is removed from the preference lists of all other agents. This removal is necessary as it prevents the same accelerator from being allocated to multiple agents and ensures that each accelerator is uniquely assigned. Subsequent to the allocation, the scheduler then reduces the budget of the agent by one.

This process is repeated iteratively, with the mechanism continuously selecting agents based on their remaining budgets and preference lists. The iteration continues until there are no agents left with a positive budget who desire any accelerator above their threshold

values. This condition signifies the completion of the allocation round, ensuring that all available resources are distributed among the agents in accordance with their preferences, budgets, and the utility thresholds they have set.

Algorithm 1 Most-Token-First Scheduler

Input: Preferences P^r, Thresholds T^r, Budgets B^r, Last turn of RR *turn*, Agents' weights W
$assignemnts = list()$
$tokens = 0$
for $i = 1$ **to** m **do**
  append *null* to *assignments*
end for
while true **do**
  a = agent with max budget and available needed accelerator w.r.t. P^r, T^r, B^r
  if a is *null* **then**
    break
  end if
  $accelerator$ = get first item from $P^r[a]$
  $assignments[accelerator] = a$
  Remove *accelerator* from P^r
  $B^r[a] := B^r[a] - 1$
  $tokens := tokens + 1$
end while
$DistributeTokens(tokens, W)$
while at least one accelerator remains **do**
  Assign remaining accelerator w.r.t. P^r and *turn*
  Update *turn*
end while

4.1.4 Complexity

The computational complexity of the Most-Token-First (MTF) algorithm, a critical aspect of its efficiency, can be methodically analyzed in terms of its dependence on the number of accelerators and agents involved. Let m represent the total number of accelerators available for allocation, and n denote the number of agents participating in the allocation process. During each iteration of the algorithm, a key step involves identifying the agent with the highest budget. This step requires a comprehensive comparison among all n agents to determine the one with the maximum available budget. Consequently, this identification process necessitates $O(n)$ operations, as it involves a linear scan through the list of agents. Once the agent with the highest budget is identified, the next step is to assign a accelerator to this agent. This assignment is relatively straightforward and can be achieved in constant time, denoted as $O(1)$. The simplicity of this step stems from the fact that the agent's top preference, which meets their utility threshold, is readily accessible from their ordered preference list. Following the assignment of the accelerator, the algorithm then proceeds to remove the allocated accelerator from the preference lists of all other agents. The process of removing a accelerator from each agent's preference list involves iterating over all n agents, resulting in a complexity of $O(n)$ for this step. Considering that this procedure is repeated for each of the m accelerators, the overall complexity of the MTF algorithm accumulates. This repetitive nature of the algorithm leads to a total complexity of $O(m \times n)$, as the procedure is conducted once for each accelerator.

In the worst-case scenario, every single accelerator is allocated following this exact procedure, reinforcing the worst-case complexity of $O(m \times n)$. However, in the best-case scenario, where no agent has preferences with utilities exceeding their thresholds, the performance of the MTF algorithm parallels that of a round-robin algorithm. In such a scenario, each agent is allocated a accelerator in a sequential manner without the need for extensive preference and budget comparisons Therefore, in this situation, the complexity is $O(m)$.

4.2 Learning

As we mentioned before, in our resource allocation system, each agent needs to report their preferences to the scheduler. This reporting is essential as it enables the scheduler to allocate accelerators to agents in a manner that aligns with their preferences. Thus, it can optimizes the utility that each agent derives from the allocation. In the construction of the preference list P_i^r, it is imperative for an agent to methodically prioritize accelerators based on the utility they offer. Specifically, an agent should position a accelerator c that provides a higher utility above other accelerators in their preference list. This prioritization is crucial due to the implications it has on the allocation outcome. If an agent were to erroneously place a less preferred accelerator c' (with lower utility) before a more preferred accelerator c (with higher utility), they might end up being allocated accelerator c' instead of c. Such an allocation would result in the agent incurring the same cost as they would have for obtaining the more preferred accelerator c, but with a lower utility payoff. This scenario is not optimal for the agent, as it leads to a suboptimal use of their resources with reduced satisfaction. Therefore, it is in the agent's best interest to meticulously rank the accelerators in their preference list such that higher utility accelerators are positioned above those with lower utility.

Agents must give the scheduler with their utility threshold tau in addition to expressing their preferences. This threshold plays an important role in the allocation process, particularly in guiding the scheduler's decisions regarding whether an agent should expend a token for an assigned accelerator. The determination of this utility threshold τ in each round is a significant decision that the agents must make. It is not a trivial task and requires careful consideration, as it directly influences the cost-benefit analysis associated with each potential accelerator allocation. Agents are required to set this threshold based on their prior experiences and observations within their respective workloads. This decision-making process demands a keen understanding of the value and utility derived from various accelerators in past allocation rounds.

The complexity of this setting may not be immediately evident. It involves a multifaceted evaluation of past allocations, utility received, and the changing dynamics of the agents' workloads. Therefore, agents require a robust learning mechanism to effectively determine

the optimal value of τ for each round. This learning mechanism is essential for agents to adapt to the evolving allocation landscape and to make informed decisions that optimize their utility over time. To address this challenge of learning the optimal threshold value, our system incorporates two distinct types of learning methods: no-regret [20] learning and actor-critic [39] learning. No-regret learning focuses on minimizing regret over a series of decisions, enabling agents to learn from past allocations and adjust their strategies accordingly. On the other hand, actor-critic learning, a reinforcement learning approach, involves agents (actors) making decisions and learning from the feedback (critiques) on the outcomes of those decisions.

Through a comprehensive evaluation process, we aim to select the learning method that yields the most superior results in terms of optimizing the agents' utility and enhancing the overall efficiency of the allocation process. This evaluation is crucial as it determines the effectiveness of the learning mechanisms in guiding the agents to set appropriate utility thresholds. The selected learning method will be instrumental in enabling agents to adaptively set their thresholds in response to the dynamic conditions of each allocation round, thereby optimizing their participation in the resource allocation process.

4.2.1 No-Regret Algorithm

No-regret learning [20] is an iterative learning process employed by agents in our system, where the objective is to minimize regret over time. Regret in this context is defined as the difference in performance between the agent's chosen strategy and the best possible strategy that could have been followed. In our approach, the strategies are represented by the choice of utility thresholds that agents can report. These thresholds are essentially the *experts* that the agent compares itself against to evaluate its performance. However, given the practical limitations of computational resources and the need for manageability, it is not feasible to consider an infinite number of possible thresholds. Therefore, we discretize the range of possible thresholds into a finite set for operational efficiency. This range is denoted by $U = \{0.0, 0.1, ..., 0.9\}$, providing a comprehensive but finite set of threshold values that agents can choose from. Additionally, in our system, we make an important assumption regarding the utilities provided by the accelerators. We assume that these utilities are

normalized within the range of $[0, 1)$. The method that we use for this normalization is mentioned in section 5.2.

In each step of the learning process, the agent employs the Hedge algorithm [20], to select a threshold from the set U. The Hedge algorithm is a well-established method in the realm of no-regret learning. Upon selecting a threshold, the agent then proceeds to update the weights associated with all possible thresholds in the set U. This update is based on the rewards received by the agent, which are contingent on the performance of the chosen threshold in the current allocation round.

The reward for a particular threshold τ for agent i in round r is quantified through a specific equation:

$$R_i^{\tau,r} = \left(\sum_{c \in C_i^{\tau,r}} u_i^{c,\tau,r} \right) - \mathbb{E}[u_i^r] * B_i^{r+1} \tag{4.2}$$

where, $C_i^{\tau,r}$ represents the collection of accelerators allocated to agent i in round r as a result of selecting the threshold τ. The term u_i^r denotes the array of utility values that agent i can potentially derive from all accelerators during round r. Additionally, B_i^r signifies the budget available to agent i for that particular round. This equation is designed to capture the efficacy of the threshold τ in terms of the utility gained by the agent and the cost incurred. The reward calculation is a crucial component of the learning process, as it directly influences the agent's future threshold selections by adjusting the weights based on the observed outcomes.

In our system, the state of an agent is characterized by their current budget. However, the Hedge algorithm, which is integral to our learning mechanism, does not inherently accommodate direct input regarding the agent's budget. To navigate this limitation, we introduce a unique learner for each possible state of an agent's budget, denoted as B (the total number of tokens in the system). This approach, while necessary, leads to a significant increase in the complexity of the learning process. Algorithm 2 shows the procedure of this method.

The intricate nature of this learning mechanism is further compounded by the fact that the computational complexity of the Hedge algorithm is expressed as $|B|^{|U|}$. It is

Algorithm 2 No regret algorithm

$w = list()$
$loss = list()$
for $i = 1$ **to** $|B_{total}|$ **do**
  $w[i] = 1$
  $loss[i] = 0$
end for
while *cnovergence OR maximum round passed* **do**
  Choose one threshold based on w
  $\bar{u}^{r'} = best\ utility\ in\ curr\ round$
  $loss[i] = \frac{1}{r} \sum_{r'=1}^{r} u_i^{r'} - \bar{u}^{r'}$
  $w[i] := w[i] \cdot exp(\epsilon \cdot loss[i])$
end while

the number of experts that an agent uses totally when it learns. Here, $|U|$ signifies the number of thresholds in the finite set of possible thresholds. This formulation indicates that the complexity escalates exponentially with both the number of thresholds and the varying states of the budget. Unfortunately, our system faces constraints in reducing the number of budget states, which are intrinsic to the nature of our resource allocation process. Consequently, this leaves us with the alternative of modifying U, the set of possible thresholds, as a means to manage and potentially reduce the overall complexity. This complexity notably impacts the speed of learning, as a more complex algorithm requires more computational resources and time to converge to an effective decision-making strategy. In addition to these challenges, we observe that the no-regret learning approach demonstrates lower performance compared to the actor-critic model. This observation is further elaborated and demonstrated in the following section of our analysis.

4.2.2 Actor-Critic Method

The Actor-Critic [39] method is a widely recognized approach in the field of reinforcement learning, distinguished by its synthesis of policy-based and value-based methodologies. This

hybrid structure facilitates a more comprehensive and effective learning process for agents operating within a given environment [6, 25]. To elucidate the distinct components of this method, we explain more detailed examination of both the *Actor* and the *Critic* aspects in the subsequent paragraphs.

The *Actor* component in the Actor-Critic framework embodies the policy aspect of reinforcement learning [39]. Essentially, it describes the strategy or behavioral pattern that an agent adopts for selecting actions in response to the prevailing state of the environment. Within the context of our system, the action taken by an agent is manifested in the form of selecting a threshold value that the agent reports in each allocation round. Additionally, the primary objective of the agent is to maximize its utility, which is achieved through the outcomes of its actions. The utility gained serves as the reward for the agent, motivating the actor to learn and adopt the most advantageous threshold selection strategy. This reward for agent i in round r, denoted as R_i^r, is quantified as the cumulative utility derived from all accelerators assigned to the agent:

$$R_i^r = \sum_{c \in C_i^r} u_i^{c,r} \tag{4.3}$$

where C_i^r represents the set of accelerators allocated to agent i in round r, and $u_i^{c,r}$ denotes the utility of each accelerator c to the agent.

The *Critic* aspect of the method focuses on estimating and evaluating the value function [39]. The critic's primary function is to assess and provide feedback on the actions executed by the actor. This feedback mechanism is pivotal as it assigns a value to each state encountered by the agent. It enables the actor to measure the effectiveness of its actions. The critic's role is vital in the learning process, as it offers a perspective on the quality and potential outcomes of the actions taken by the actor. Subsequently, the actor incorporates this feedback to refine and adjust its policy. By using this feedback, actor enhances the likelihood of achieving higher rewards. This iterative process of feedback and adjustment is central to the Actor-Critic method, where the critic informs the actor about the efficacy of its actions, and the actor utilizes this information to optimize its policy for better future performance.

The state of each agent is defined by a specific set of parameters. This state is defined

as $s_i^r = \{B_i^r, u_i^r\}$, where each element represents a key factor in the agent's decision-making process. The budget of the agent at round r is represented by B_i^r. The second element, u_i^r, signifies the utilities of accelerators for the agent. To update the estimated value of the current state based on the observed reward and the estimated value of the next state, the following equation is employed:

$$V(s_i^r) := V(s_i^r) + \alpha(R_i^r + \gamma V(s_i^{r+1}) - V(s_i^r)) \tag{4.4}$$

In this equation, $\alpha \in (0, 1]$ represents the learning rate, which controls the rate at which new information overrides old information. The term $\gamma \in (0, 1]$ serves as the discount factor for future values, indicating the importance of future rewards compared to immediate rewards. The function $V(\cdot)$ denotes the value function, which is learned during the process for each state. This function estimates the expected utility or benefit of being in a certain state. It considers both immediate and future rewards.

Our system incorporates two separate neural networks for the actor and critic components. The actor network is responsible for mapping the current state of an agent to a set of probabilities for threshold selection. Meanwhile, the critic network maps the current state to its estimated value. To train both the actor and critic networks, we utilize the temporal difference (TD) learning method [69]. TD learning is a method that involves calculating the difference between the estimated outcomes (predicted by the value function) and the actual outcomes observed. This TD error is then used for backpropagation within our neural networks and allows them to learn and improve their predictions and decisions over time.

The TD error equation, which is fundamental to this learning process, can be expressed for agent i in round r as follows:

$$\delta = R_i^r + \gamma V(S_i^{r+1}) - V(S_i^r) \tag{4.5}$$

Figure 4.1 shows the procedure of working of this our method.

For the critic component, which is responsible for predicting the value associated with a given state, δ is directly applicable as an error term to facilitate its updates. To optimize the critic's learning through gradient descent, a common approach in machine learning, we

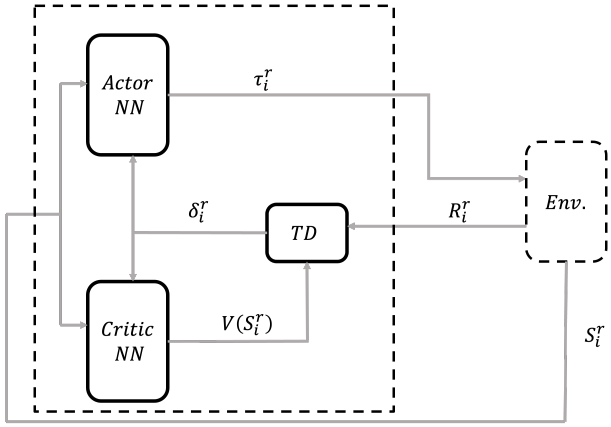


Figure 4.1: Procedure of actor critic method

utilize the square of the TD error. So, the loss function for the critic, denoted as $loss_{critic}$, is thus formulated as follows:

$$loss_{critic} = \delta^2 \tag{4.6}$$

On the other hand, the actor component requires a different approach to calculate its loss. The actor's loss is determined by considering the probability of selecting a particular threshold τ given the agent's state s_i^r, combined with the TD error δ. The loss function for the actor is formulated as follows:

$$loss_{actor} = -\log(\pi(\tau|s_i^r)) \cdot \delta \tag{4.7}$$

In this equation, $\pi(\tau|s_i^r)$ represents the probability of choosing threshold τ conditioned on the agent's current state s_i^r. The use of the logarithm of this probability, multiplied by the TD error, is a standard approach in reinforcement learning for policy gradient methods.

Hyperparameters

Hyperparameters play an important role in the configuration of machine learning models. These parameters are essential for the optimal performance of the learning algorithms [5, 17]. To efficiently search for and optimize these hyperparameters, we employ Optuna [1].

Optuna is a robust hyperparameter optimization framework known for its effectiveness in identifying the most suitable hyperparameter settings for a given model. Its capabilities are particularly beneficial in our context, where fine-tuning these parameters is key to achieving high performance. One of the notable strengths of Optuna is its ability to identify the best hyperparameter settings through an efficient search process. This process involves exploring a wide range of possible hyperparameter configurations and identifying those that yield the most promising results. Additionally, Optuna possesses the capability to prune ineffective trials, which significantly accelerates the optimization process. By eliminating trials that are less likely to produce optimal outcomes early in the process, Optuna ensures a more focused and faster search for the best hyperparameters.

Chapter 5

Evaluation

In this chapter, we examine our system. To set the stage for this evaluation, section 5.1 introduce the baselines that serve as our reference points for comparing and assessing our system's effectiveness. Following the introduction of the baselines, the subsequent section delve into an exploration of our workloads. Moving further into our evaluation, section 5.3 undertake a detailed examination of various facets of our system's performance. section 5.4 explore the overhead of scheduling within our system. Then, section 5.5 conduct a fairness analysis to ensure that our system operates equitably and does not exhibit bias in its decision-making processes.

5.1 Baseline

For our baseline, we compare our system with some state-of-the-art algorithm. Specifically, we implement the following two policies for scheduling in a heterogenous system.

- **Themis** [53]: In this method, the objective is to minimize the maximum values of $\rho = \frac{T_{sh}}{T_{id}}$ in each round. T_{sh} represents the completion time of jobs in in shared usage of accelerators. On the other hand, T_{id} denotes the completion time of jobs running on $\frac{1}{n}$ share of all accelerators.

Table 5.1: Gemmini throughputs for different computation clock frequencies

Applicaiton	Gemmini 2GHz	Gemmini 3GHz
Resnet conv	70.262	86.332
Resnet matmul	3.001	4.139
Mobilenet conv	81.644	111.995
Mobilenet WS	1.480	2.101

- **Gandiva$_{\text{fair}}$** [9]: Gandiva$_{\text{fair}}$ uses stride scheduling. In stride scheduling, each agent is assigned a "stride" value on each resource based on its weight (tickets in Gandiva$_{\text{fair}}$), which represents its priority or share of the resource time. The scheduler maintains a queue of jobs and selects the process with the smallest stride value to run next. As jobs run, their stride values are adjusted to ensure proportional allocation of accelerator time based on their assigned shares. This approach ensures fairness by dynamically adjusting priorities, allowing agents with smaller strides to receive more resource time.

5.2 Workloads

The tasks in our setup involve a combination of multiple neural network inference processes that run on the Gemmini [22] accelerator at various clock frequencies. We specify the amount of inference that each architecture can provide to different applications, and these are detailed in Table 5.1. In our setup, the total number of inferences is the throughput for each agent.

To manage the distribution of these tasks across different accelerators, each agent is equipped with a Power-of-Two-Choice load balancer. This load balancer handles task assignment, ensuring that tasks are evenly distributed among the available resources. Jobs have a deadline, meaning they will be dropped from the queue after passing their deadlines. Furthermore, we assume that the arrival rate of jobs follows a Poisson distribution with a

$\lambda = 40$. This is the base parameter for arriving rates. Each agent, based on his weight, has a coefficient that specifies its own λ_i.

During each iteration, agents submit their demand for the resources they require, and each resource can only process tasks that are present in its queue. To assess the performance of our agent policies, we conducted a benchmark that spanned 300 iterations, providing insights into how effectively our policies operate in this context.

In our system, we have implemented a strategy where each agent is associated with a worker responsible for managing the runtime operations of the agents. These workers are tasked with executing the procedures required by each agent. This approach is chosen because the tasks undertaken by each agent are entirely independent of one another. While it is possible to assign each agent to a single thread, we found that this incurred a significant overhead for our system. Therefore, to streamline operations, we have distributed agents among these workers, with each worker dedicated to handling the tasks of a specific agent.

The scheduler is the part of a component that is called coordinator. The coordinator facilitates communication with the workers, exchanging data with them and ensuring that the necessary information is relayed to each worker. To facilitate this communication and maintain synchronization between the coordinator and workers, we have implemented a message queuing system. The overview of our system is shown in Figure 5.1.

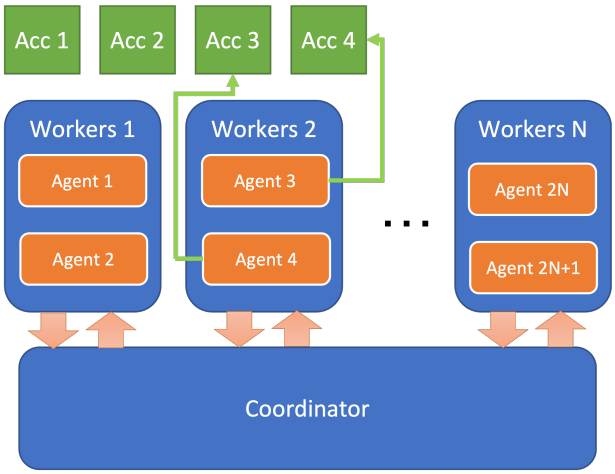


Figure 5.1: Overview of architecture of implementation.

In our system, there are twenty agents sharing forty accelerators, and these accelerators are heterogeneous in nature, offering diverse capabilities and performance characteristics. It allows us to evaluate and adapt our approach to various scenarios and resource configurations.

5.3 Performance

To assess how well our system performs, we analyze it in two scenarios. First, we examine its performance when all agents have identical weights. Then, we explore a scenario where agents possess varying weights. In both cases, we ensure that agents' queue utilities, which represent the ratio of their arrivals to departures based on their weight, remain constant. We vary these queue utilities from 40% to 80% in our experiments to observe the system's behavior under different loads.

In Figure 5.2, we present the weighted social welfare in the scenario where all agents have the same weights. Social welfare is the weighted average utility that agents get through the experiment. In our system, utility is the accumulated throughput that an agent can get from the accelerators assigned to it. Figures 5.3 illustrates the average throughput of all agents within this uniform weight setup. As you can see, at 60% queue utility, our technique exceeds Themis by 152%. On average, it outperforms two other schedulers by 39%.

To explore the impact of weight heterogeneity, we devised three distinct setups. In the first setup, we divided the agents into two equal groups: half with a weight of 1 and the other half with a weight of 2. Figure 5.4 provides insight into the weighted social welfare within this configuration. Additionally, Figure 5.5 offers a detailed view of the average throughput of all agents in this particular setup. Further granularity is provided by Figure 5.6, which showcase the average throughput of each agent type categorized by their respective weights. As shown, when queue utility is 60%, our technique surpasses Themis by 183%. It outperforms two other schedulers by an average of 43%.

In the second setup, we diversified the weights of agents as follows: half of the agents retained a weight of 1, while 25% were assigned a weight of 2 and the remaining 25% were endowed with a weight of 4. The weighted social welfare for this arrangement is depicted

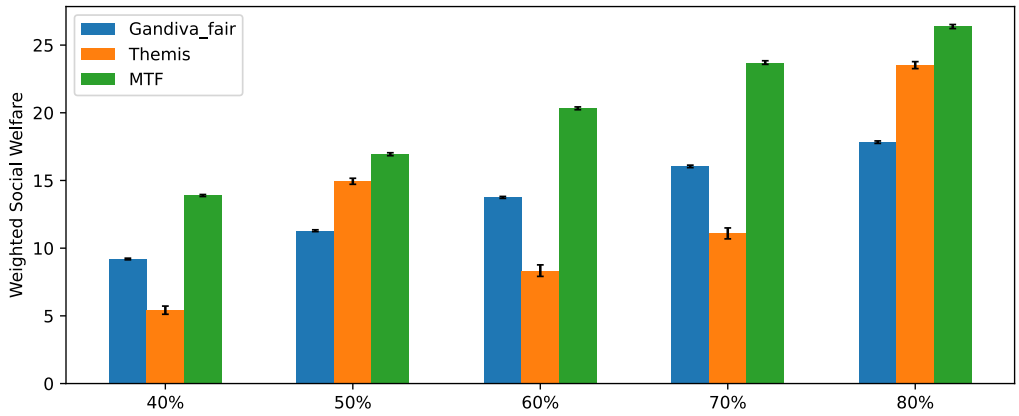


Figure 5.2: Weighted social welfare for setup that all agent has same $w = 1$.

in Figure 5.7. Moreover, Figures 5.8 presents the average throughput of all agents in this setup, offering insights into how the system handles varied weight distributions. Figures 5.9 delves deeper, presenting the average throughput of each agent type based on their individual weights. When queue utility is 70%, our technique outperforms Themis by 73%. In compared to the other two schedulers, it outperforms them by 25%.

In the third setup, we diversified the weights of agents to further examine system performance. Here, we allocated weights as follows: half of the agents retained a weight of 1, a quarter of the agents were assigned a weight of 2, and the remaining quarter boasted a weight of 8. This deliberate variation in weights allows us to explore how the system copes with a wider range of weight discrepancies. Figure 5.10 serves as a visual representation of the weighted social welfare within this particular setup. Figure 5.11 offers detailed insights into the average throughput of all agents in this setup. Figure 5.12 shows the average throughput of each agent type based on their individual weights in this setup. As you can see, our technique outperforms Themis by 62% when queue utility is 50%. When compared to two other schedulers, it performs 14% better on average.

To assess the impact of deadlines on our system, we analyze it with various deadlines for dropping jobs from the queue. Figure 5.13 illustrates the effects of different deadlines on

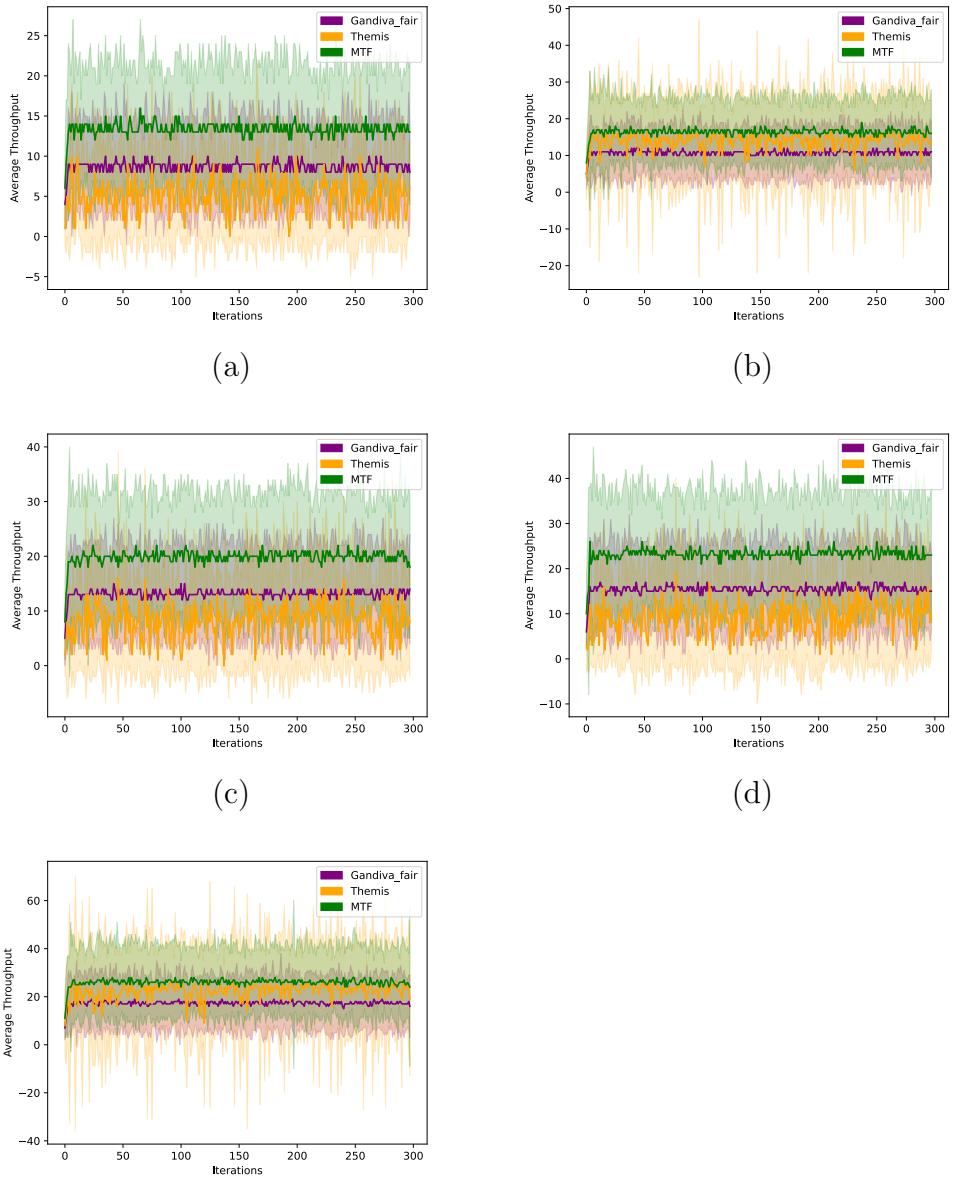


Figure 5.3: Average Throughput for setup all agent has same weights and queue utility is (a) 40% (b) 50% (c) 60% (d) 70% (e) 80%.

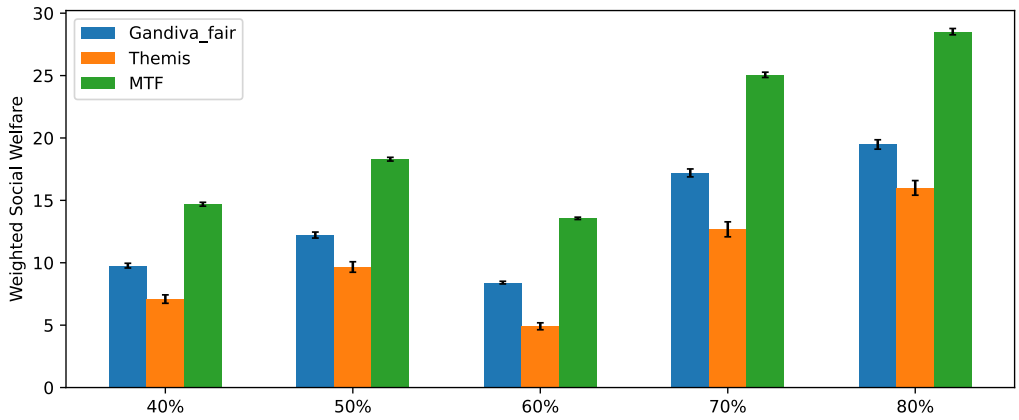


Figure 5.4: Weighted social welfare for setup that half agents have $w = 1$ and other half have $w = 2$.

our system. The performance of the Themis cannot improve after certain deadline, however we can get the best performance that can be reached using $\text{Gandiva}_{\text{fair}}$. In average, we have 8% improvement in comparison with $\text{Gandiva}_{\text{fair}}$, and in overall, we outperform 13% to others.

To explore our system under conditions where each agent possesses a unique queue utility, we conducted experiments using two distinct setups. In the first setup, we divide the agents as follows: half of them are assigned a weight of 1 with a queue utility of 70%, 25% have a weight of 2 with a queue utility of 50%, and the remaining 25% have a weight of 4 with a queue utility of 40%. Figure 5.14 visually represents their average throughput and the throughput of each agent type within this setup. In the second setup, the agents' queue utilities are different: 50% have a weight of 1 with a queue utility of 40%, 25% have a weight of 2 with a queue utility of 50%, and the remaining 25% possess a weight of 4 with a queue utility of 70%. Figure 5.15 displays the average throughput and the throughput of each agent type within this alternative setup.

It is feasible to determine the ideal threshold for each agent in every round, but it is not practical to demonstrate this for this number of agents. Therefore, considering the performance enhancement observed for each user type (based on their weights) and the

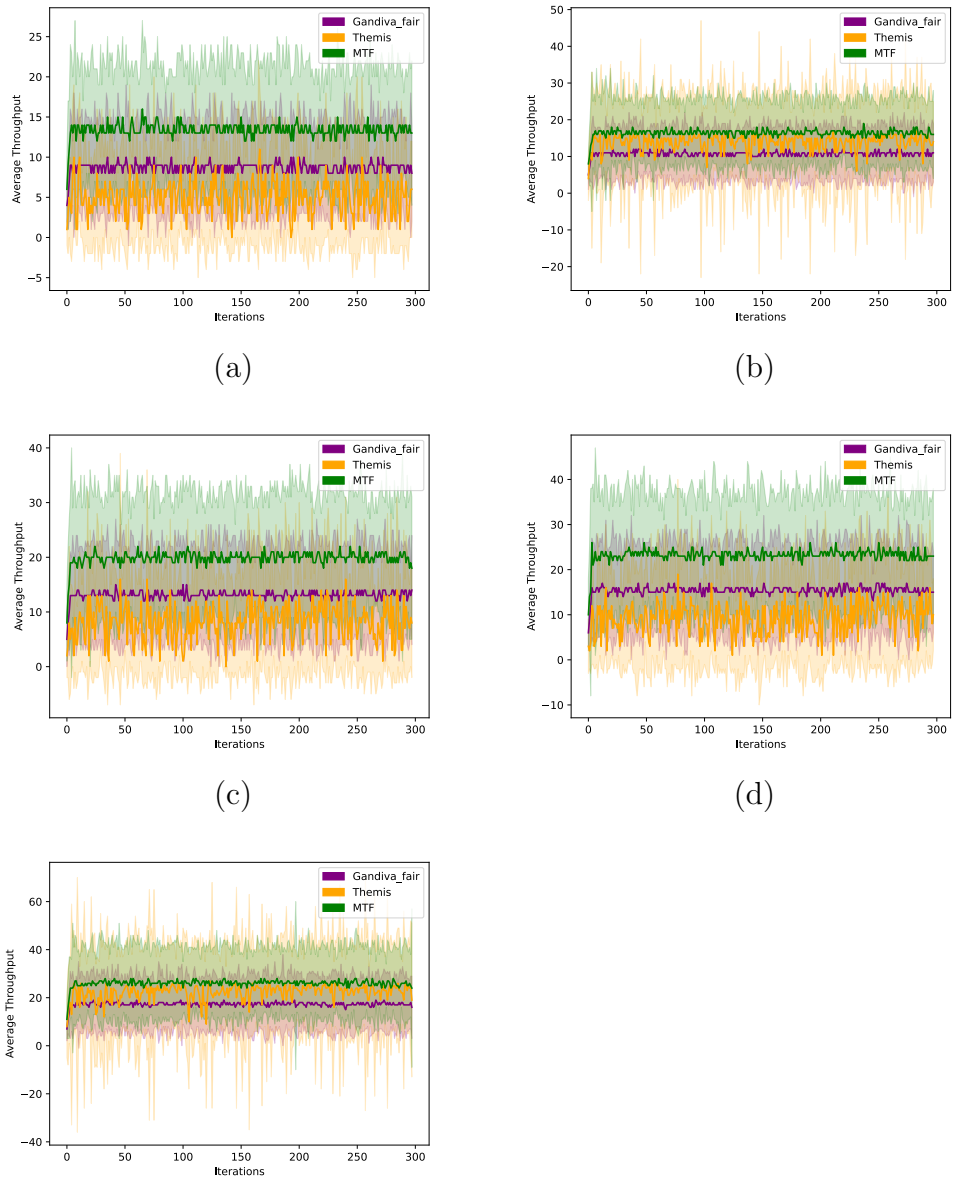
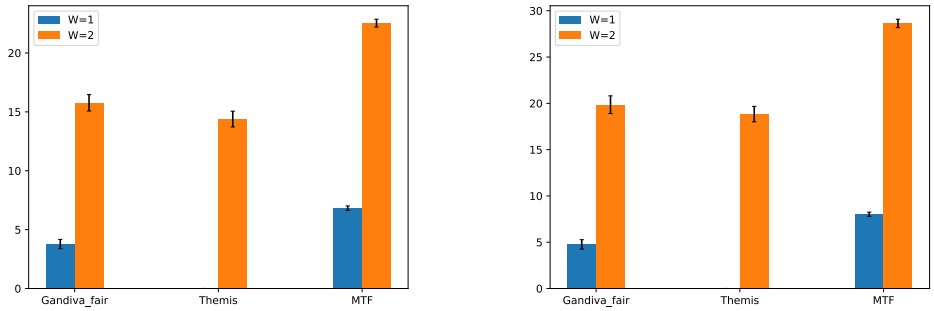
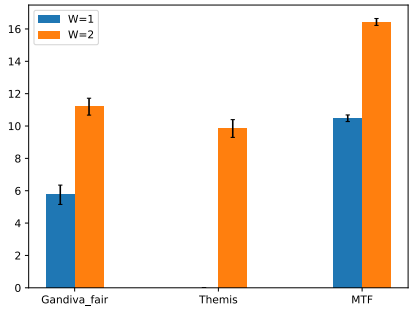


Figure 5.5: Average Throughput for setup half agents have $w = 1$ and other half have $w = 2$ and queue utility is (a) 40% (b) 50% (c) 60% (d) 70% (e) 80%.

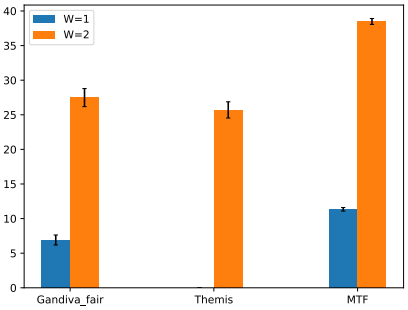


(a)

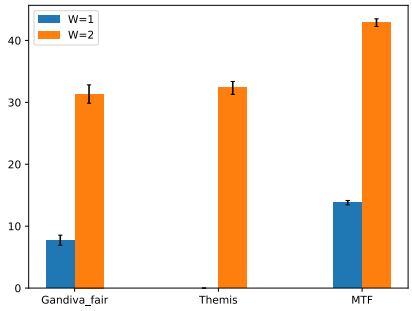
(b)



(c)



(d)



(e)

Figure 5.6: Average throughput of each type of agent for setup half agents have $w = 1$ and other half have $w = 2$ and queue utility is (a) 40% (b) 50% (c) 60% (d) 70% (e) 80%.

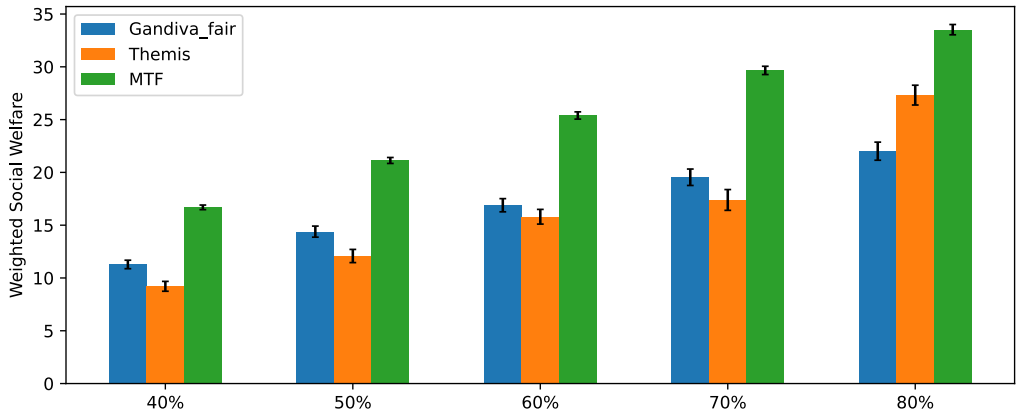


Figure 5.7: Weighted social welfare for setup that half agents have $w = 1$, 25% of agents have $w = 2$ and 25% of them have $w = 4$.

narrow confidence interval, it appears that they were all trained to select the optimal value.

5.4 Scheduling Overhead

In Figure 5.16 and Figure 5.17, we present a detailed analysis of the scheduling overhead's impact on the system's runtime. This metric measures the average time that the coordinator dedicates to the scheduling algorithm. To comprehensively explore the influence of both the number of agents and the number of accelerators on scheduling, we conducted various experiments with distinct setups.

First, to understand the effect of the number of agents, we maintained a constant number of accelerators at 40 while varying the number of agents from 20 to 60 and we show it in Figure 5.16. To investigate the impact of the number of accelerators, we fixed the number of agents at 20 and varied the number of accelerators from 20 to 60 and it is demonstrated in Figure 5.17.

The Themis scheduling algorithm incurs the highest scheduling overhead because it involves solving an Integer Linear Programming (ILP) problem, which is known for its

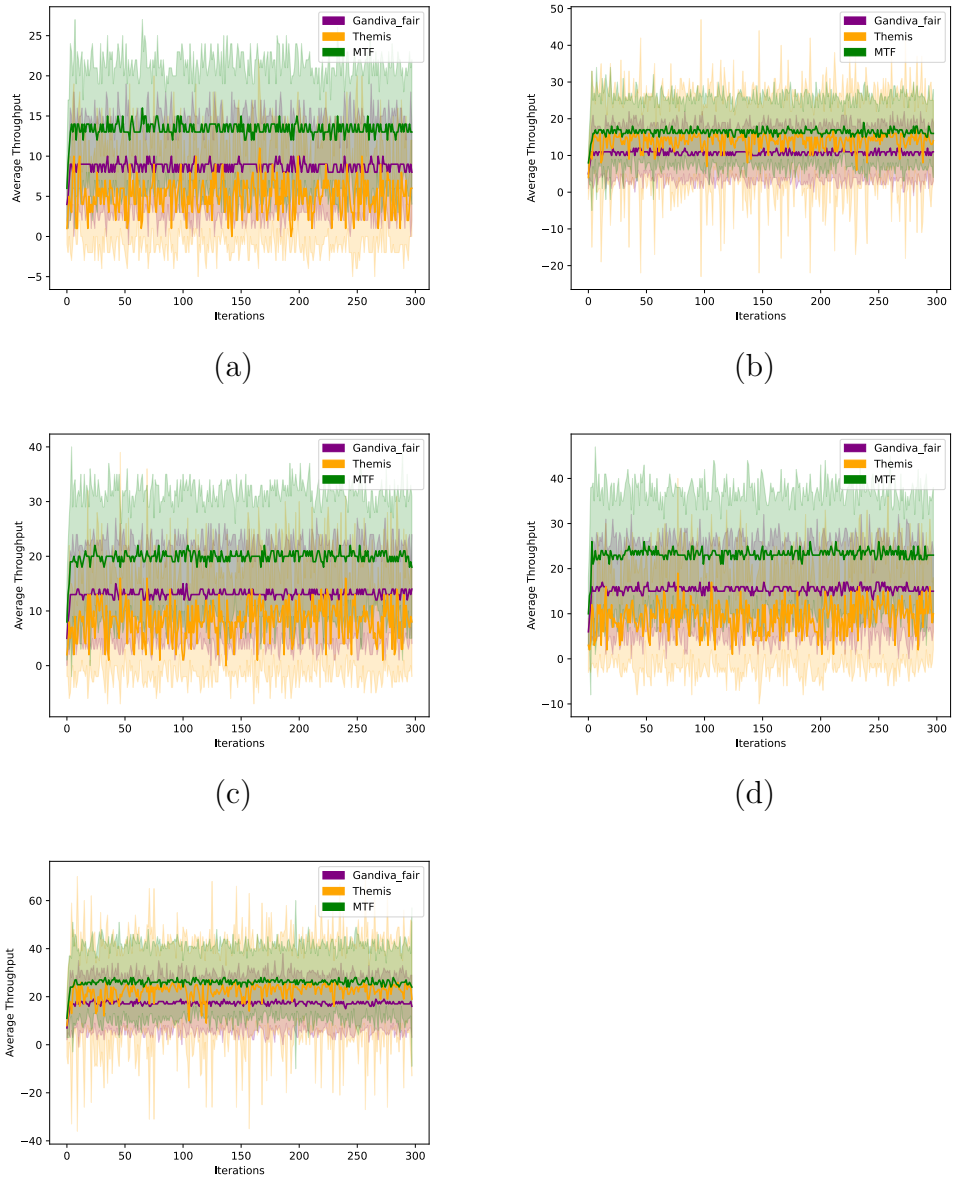
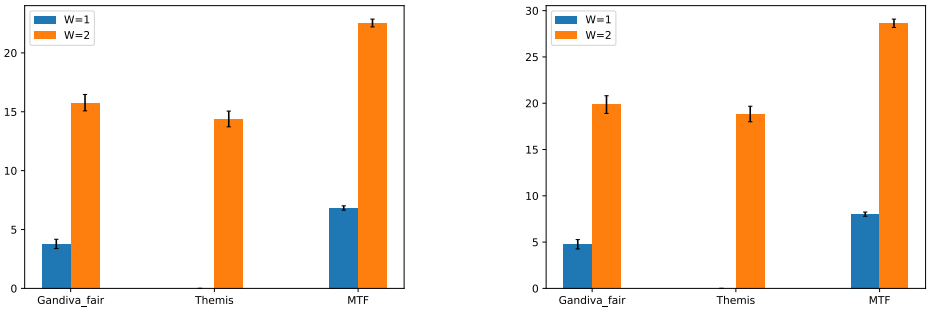
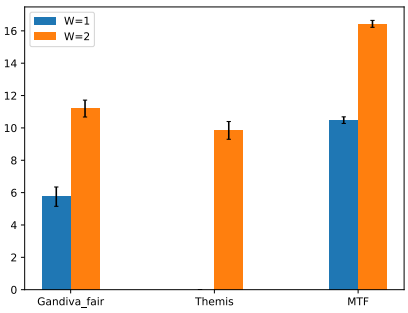


Figure 5.8: Average Throughput for setup half agents have $w = 1$, 25% of agents have $w = 2$ and 25% of them have $w = 4$ and queue utility is (a) 40% (b) 50% (c) 60% (d) 70% (e) 80%.

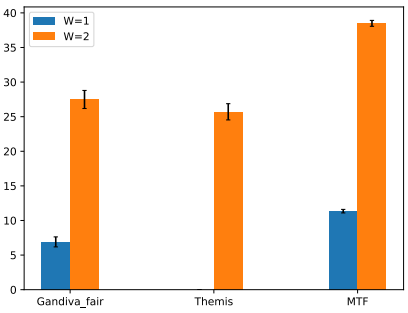


(a)

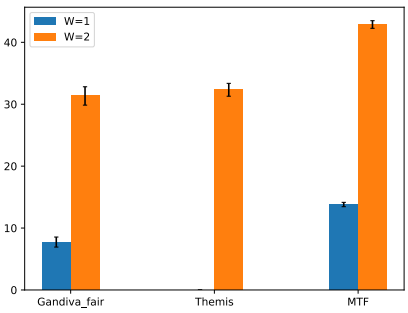
(b)



(c)



(d)



(e)

Figure 5.9: Average throughput of each type of agent for setup half agents have $w = 1$, 25% of agents have $w = 2$ and 25% of them have $w = 4$ and queue utility is (a) 40% (b) 50% (c) 60% (d) 70% (e) 80%.

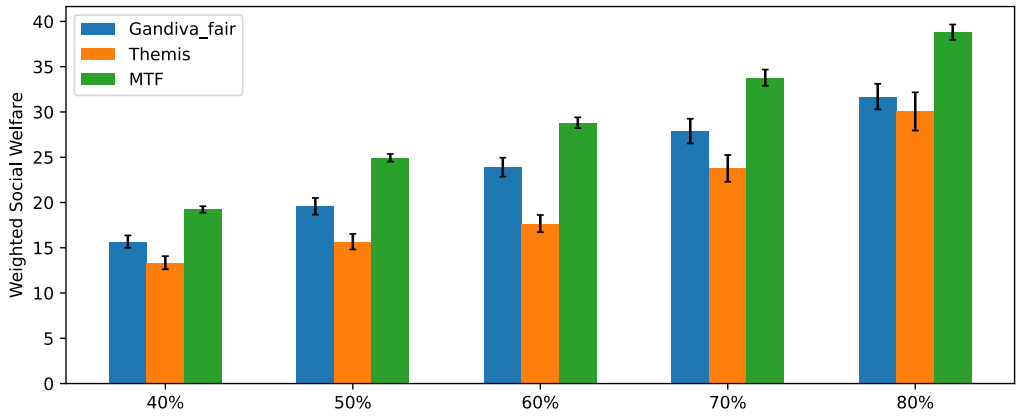


Figure 5.10: Weighted social welfare for setup that half agents have $w = 1$, 25% of agents have $w = 2$ and 25% of them have $w = 8$.

exponential complexity. Comparing our system's performance with that of Gandiva$_{\text{fair}}$, we find that our system demonstrates superior scheduling times. This improvement showcases our system's effectiveness in reducing scheduling overhead.

5.5 Fairness Analysis

5.5.1 Sharing Incentive

In our system, we evaluate fairness using a specific metric outlined in Section 3. To show that our system adheres to the Sharing Incentive (SI) rule, we get the ϕ distribution on agents and show what is the distribution of ϕ. Figure 5.18 displays this distribution in setup when 50% of agents have $w = 1$, 25% have $w = 2$ and the remaining 25% have $w = 4$ and queue utility of all agents are the same. As you can see, our system has better distribution and satisfies most of the agents with better ϕ. Therefore, it becomes evident that our system comfortably fulfills this requirement for most of agent. Consequently, we can confidently state that our mechanism has better SI with comparison with other

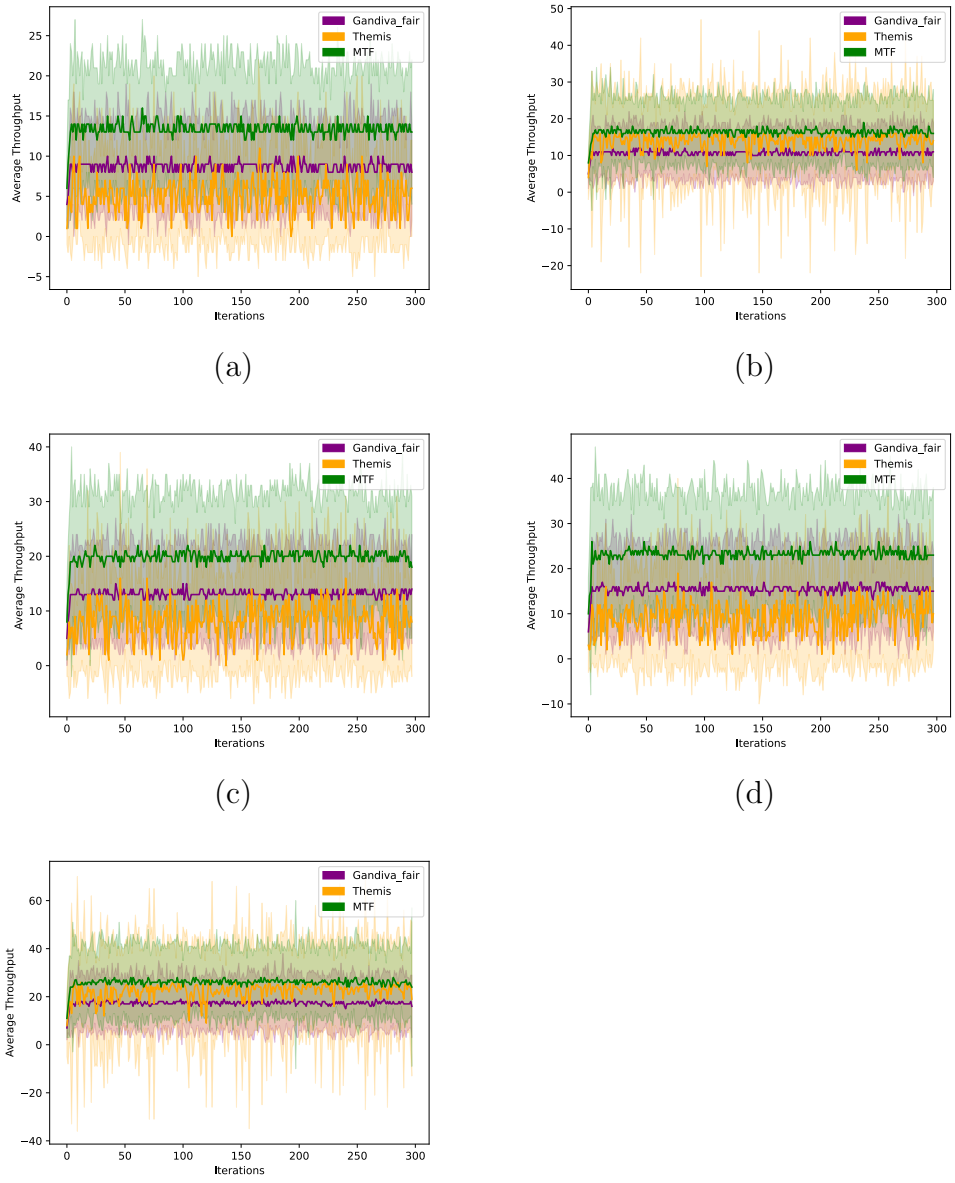
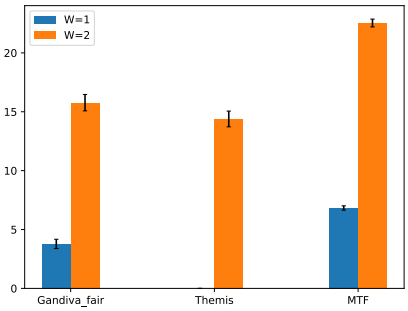
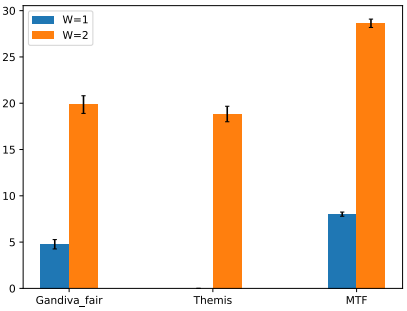


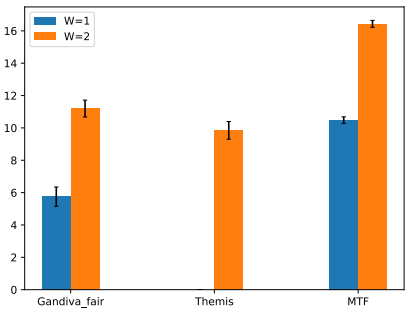
Figure 5.11: Average Throughput for setup half agents have $w = 1$, 25% of agents have $w = 2$ and 25% of them have $w = 8$ and queue utility is (a) 40% (b) 50% (c) 60% (d) 70% (e) 80%.



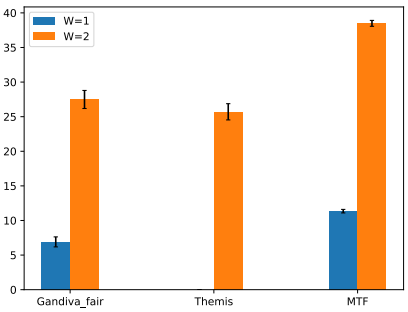
(a)



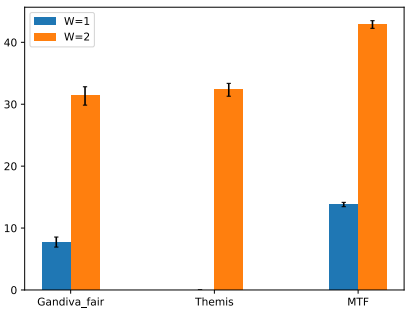
(b)



(c)



(d)



(e)

Figure 5.12: Average throughput of each type of agent for setup half agents have $w = 1$, 25% of agents have $w = 2$ and 25% of them have $w = 8$ and queue utility is (a) 40% (b) 50% (c) 60% (d) 70% (e) 80%.

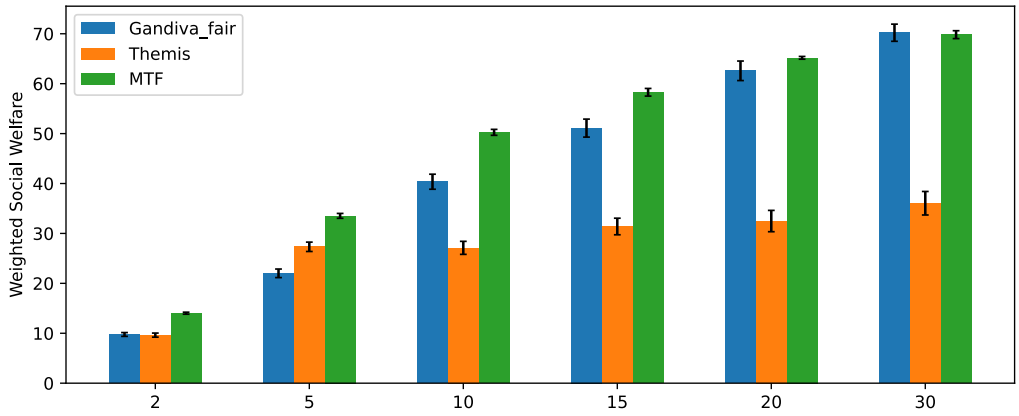
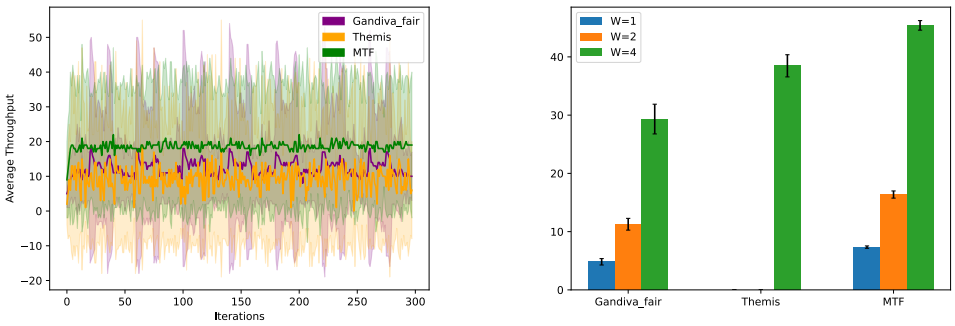
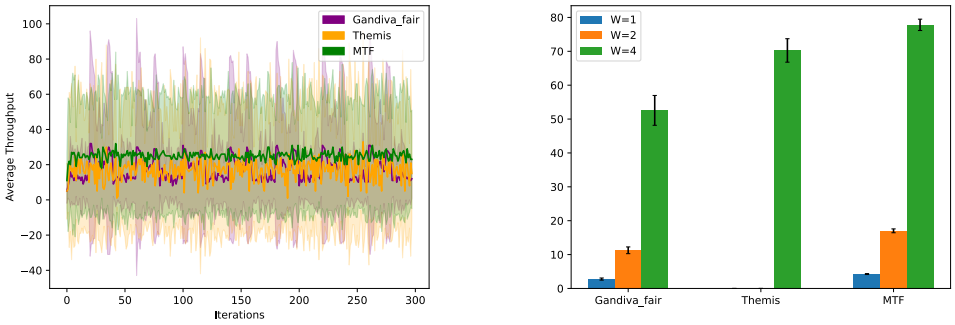


Figure 5.13: Weighted social welfare for setup that half agents have $w = 1$, 25% of agents have $w = 2$ and 25% of them have $w = 4$ with different deadlines (X-axis shows the deadline).



(a) (b)

Figure 5.14: (a) Average throughput of all agents (b) average throughput of each type of agent for setup half agents have $w = 1$, 25% of agents have $w = 2$ and 25% of them have $w = 8$ and queue utility 70% for agents with $w = 1$, 50% for agents with $w = 2$ and 40% for agents with $w = 4$.



(a) (b)

Figure 5.15: (a) Average throughput of all agents (b) average throughput of each type of agent for setup half agents have $w = 1$, 25% of agents have $w = 2$ and 25% of them have $w = 8$ and queue utility 40% for agents with $w = 1$, 50% for agents with $w = 2$ and 70% for agents with $w = 4$.

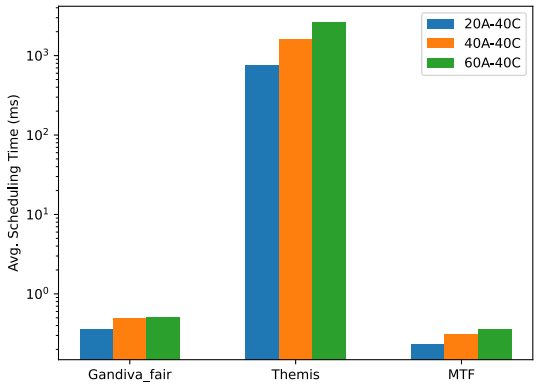


Figure 5.16: Effect of number of agents on scheduling overhead

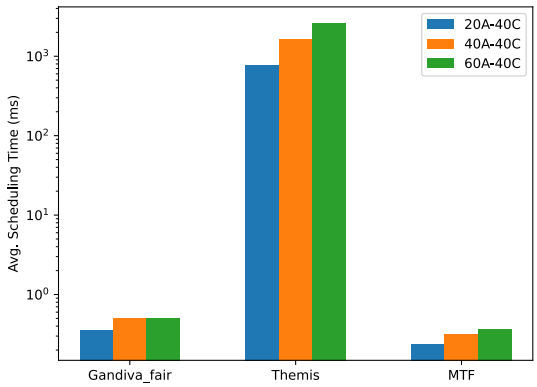


Figure 5.17: Effect of number of accelerator on scheduling overhead

algorithms, promoting fairness in how resources are allocated in our system.

5.5.2 Envy-free

To provide a comprehensive insight of EF, we present Table 5.2, which provides a detailed breakdown of the utilities experienced by individual agents in different allocation scenarios. Given constraints in illustrating scenarios involving a large number of agents, we have chosen a setting in which 10 accelerators are shared among 5 agents. All agent has same weights and queue utility which is 80%. In this table, each row represents a unique agent with distinct characteristics, and the columns indicate the allocations they receive. All agents achieve their highest utility when they are assigned their designated allocations. This observation suggests that none of the agents desire allocations other than their own. This innate contentment, where no agent covets the allocations of others, highlights the satisfaction of the envy-freeness (EF) constraint.

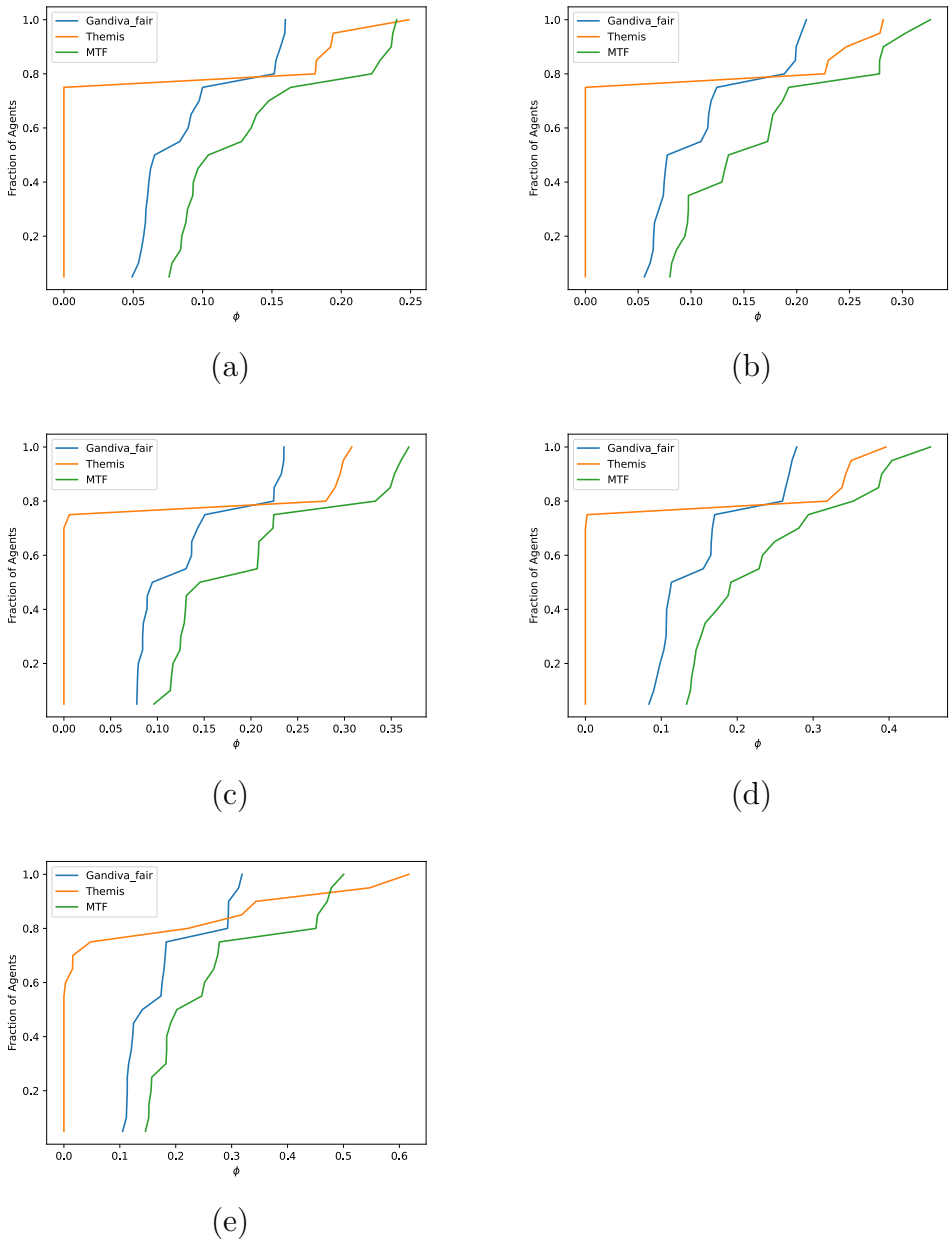


Figure 5.18: Sharing Incentive index for setup that half of agents have $w = 1$, 25% have $w = 2$ and 25% have $w = 4$ and queue utility is (a) 40% (b) 50% (c) 60% (d) 70% (e) 80%.

Table 5.2: Alteration utilities with other agents' allocations

Agents	A_1 Alloc	A_2 Alloc	A_3 Alloc	A_4 Alloc	A_5 Alloc
u_1	**74.29**	72.18	62.09	72.42	73.02
u_2	63.18	**74.00**	62.32	72.23	66.44
u_3	65.77	70.59	**74.23**	72.84	67.51
u_4	65.68	68.45	66.83	**75.19**	62.60
u_5	68.79	71.56	56.76	67.54	**73.32**

Chapter 6

Related Works

6.1 ML Jobs Schedulings

A number of studies have been conducted to address this need, each proposing various frameworks and methodologies. One of the seminal works in this domain is presented by [76]. They introduce an innovative cluster scheduling framework named Gandiva. The framework leverages specialized knowledge in the domain of deep learning to enhance the speed and efficiency of training deep learning models within a GPU cluster. Gandiva's approach is unique in its ability to dynamically adapt to the varying requirements of deep learning tasks, ensuring optimal utilization of GPU resources.

Concurrently, [26] approach the problem from a slightly different angle. They propose a GPU cluster manager, named Tiresias, which is specifically customized for distributed deep learning training tasks. Tiresias effectively arranges and positions deep learning tasks within a GPU cluster to significantly reduce their job completion times (JCTs). This approach is particularly notable for its predictive job scheduling algorithm, which anticipates the resource needs of different tasks.

Further contributions to this field come from Optimus [56] and Salus [80], who each devise their own solutions to optimize the scheduling and management of GPU resources for deep learning. While Optimus focuses on dynamic resource allocation strategies, Salus

introduces a novel way of memory management to reduce the overhead of context switching in GPU clusters. However, these works present two notable shortcomings. Firstly, they predominantly focus on deep learning workloads. This is a limitation considering the wide variety of machine learning applications that exist globally, each with its unique computational requirements. Secondly, their primary consideration revolves around performance enhancement. While performance is undoubtedly crucial, this singular focus often comes at the expense of other critical factors such as fairness and resource allocation equity.

Recognizing these gaps, newer research works have emerged. Studies like Gandiva$_{\text{fair}}$ [9], Themis [53], AntMan [77], and TGS [75] have made strides in addressing fairness issues within their systems. These efforts are commendable as they attempt to balance both fairness and performance concerns in GPU cluster scheduling. However, they still remain predominantly centered on the optimization of deep learning workloads. In contrast, our approach diverges from the existing literature by being applicable to a broader spectrum of applications. This makes our method more versatile and inclusive, catering to the diverse landscape of machine learning applications beyond just deep learning. We believe that this comprehensive approach not only enhances performance but also ensures a more equitable distribution of resources among various machine learning tasks.

6.2 Heterogeneous System Scheduling

Several studies have endeavored to develop scheduling techniques that can efficiently utilize the diverse resources of these systems. These methodologies range from sophisticated mathematical models to heuristic approaches, each aiming to optimize various aspects of system performance.

Chen proposes a novel method that involves the projection of core configurations and program resource demands into a multi-dimensional space in [10]. This approach is guided by weighted Euclidean distances, which help in accurately mapping tasks to the most suitable cores based on their computational needs and the core's capabilities. This methodology is particularly effective in systems with a wide variety of core types and performance levels.

Another significant contribution is from [40], who introduce the concept of bias scheduling.

This method is tailored for systems with diverse core microarchitectures and performance characteristics. It involves identifying key metrics that can determine an application's bias towards certain core types. By doing so, the scheduler can optimize the allocation of tasks to cores that best fit their performance profile, thereby improving overall system efficiency.

Van Craeynest, in [70], introduces an innovative approach known as Performance Impact Estimation (PIE). PIE is designed to predict optimal workload-to-core mappings by estimating the performance impact of different core allocations. This technique allows for more informed scheduling decisions, leading to improved performance in heterogeneous systems.

Notably, a common theme among these schedulers is their prioritization of performance over other factors like fairness. While performance optimization is crucial, focusing solely on this aspect can lead to resource allocation disparities, especially in environments with diverse user needs and task requirements. In contrast to these existing methods, our work seeks to strike a balance between performance and fairness in the scheduling for heterogeneous systems. We propose a novel framework that not only optimizes performance but also ensures a fair distribution of resources among different tasks and users.

A notable work in this area is by [81], who employ a token-based mechanism to allocate heterogeneous processors. In their model, the allocation problem is formulated as a repeated game, where each agent (or task) is allocated tokens that they can use to bid for processor time. However, their approach assumes uniformity in the utility provided by all super cores to the agents. This assumption may not hold in real-world scenarios where different tasks may derive varying levels of utility from the same core. Our system addresses this assumption by accommodating the possibility that each cluster may provide distinct levels of utility to different agents. We introduce a dynamic utility model that adapts to the changing needs and preferences of tasks, thereby ensuring a more equitable and efficient allocation of resources in heterogeneous systems.

6.3 Token-Based Algorithms in Resource Sharing

Token-based algorithms have emerged as a popular solution in various sectors to address resource sharing. These algorithms typically involve the use of tokens as a means of incentivizing cooperative behavior, ensuring fair resource distribution, and managing user interactions effectively.

Xu develops a pioneering token system specifically designed for autonomous wireless relay networks in [78]. Their approach was aimed at encouraging cooperation within ad hoc mobile networks. In this system, tokens serve as a form of currency or incentive for self-interested users. The key idea is to encourage users to forward communication from other nodes by rewarding them with tokens. This incentivization helps overcome the inherent reluctance of users to utilize their resources for the benefit of others, thereby enhancing overall network efficiency. Similarly, Shen, in [63], examines the same scenario but approaches the problem from a slightly different angle. They create a system for token exchange to address user interference issues in ad hoc networks. In their model, tokens are utilized as a means to regulate access to the network, thereby reducing interference and contention among users. This system not only incentivizes cooperation but also ensures a more efficient utilization of network resources.

A notable aspect of both these systems is their application in decentralized systems. These token-based algorithms are particularly suited for environments where there is no central authority to regulate resource allocation or user behavior. By leveraging the concept of tokens, these systems introduce a self-regulating mechanism that promotes cooperation and efficient resource usage among independent and self-interested entities. In contrast to the aforementioned studies, our research takes a different direction by focusing on the application of token-based algorithms in the context of accelerators within centralized systems. Unlike decentralized ad hoc networks, accelerators in centralized systems are characterized by a central authority that manages resource allocation and scheduling. This presents a different set of challenges and opportunities for the implementation of token-based algorithms.

Chapter 7

Conclusion

In this thesis, we have proposed a novel token-based mechanism designed to enhance the allocation of resources in a heterogeneous accelerator environment. The fundamental premise of our approach lies in the effective communication of agent preferences to a central scheduler, which then undertakes the challenging task of optimizing resource allocation based on these preferences. To facilitate the determination of these preferences, we have leveraged the power of an Actor-Critic model, which acts as a critical component in our system, effectively establishing thresholds that guide each agent's preference reporting.

Our endeavor doesn't stop at introducing this mechanism; it also involves the systematic formalization of all essential fairness constraints. Ensuring fairness in resource allocation is a paramount concern, especially in diverse and dynamic computing environments. By integrating these fairness constraints into our model, we aim to strike a delicate balance between maximizing performance and upholding fairness principles.

An important aspect of our work involves a comprehensive analysis that pits our innovative approach against conventional resource allocation techniques. This analysis is critical in demonstrating the superior performance of our methodology across a wide range of scenarios. We have meticulously examined various performance metrics and benchmarks to establish the effectiveness and robustness of our approach. Through this analysis, we have illustrated how our method outperforms existing approaches, highlighting its potential for real-world applications.

It is important to emphasize that our approach primarily focuses on two key objectives: maximizing performance and ensuring fairness. Achieving both of these objectives simultaneously is a complex and challenging task, but it is a fundamental aspect of our work. We believe that striking the right balance between performance and fairness is vital in diverse computing landscape.

However, even as we celebrate our achievements and the promising results of our research, a critical question remains unanswered: "Is our approach the ultimate mechanism for efficiently distributing resources within heterogeneous accelerators?" This question opens the door to future research and exploration. While our approach has demonstrated improvements, the field of resource allocation in heterogeneous accelerators is ever-evolving. New technologies and methodologies may emerge, and further refinements to our approach could be possible. Therefore, ongoing inquiry and innovation in this domain are essential to ensuring that we continue to advance the state of the art in resource allocation and optimization.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[2] Danish Ali, Anees Ur Rehman, and Farhan Hassan Khan. Hardware accelerators and accelerators for machine learning. In *2022 International Conference on IT and Industrial Technologies (ICIT)*, pages 01–07, 2022.

[3] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, pages 10–21, 2020.

[4] Xiaohui Bei, Zihao Li, and Junjie Luo. Fair and efficient multi-resource allocation for cloud computing. In *International Conference on Web and Internet Economics*, pages 169–186. Springer, 2022.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 2011.

[6] Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor–critic algorithms. *Automatica*, pages 2471–2482, 2009.

[7] Yair Censor. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, pages 41–59, 1977.

[8] Dongju Chae and Parichay Kapoor. Centralized generic interfaces in hardware/software co-design for ai accelerators. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 619–622, 2020.

[9] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[10] Jian Chen and Lizy K John. Efficient program scheduling for heterogeneous multi-core processors. In *Proceedings of the 46th Annual Design Automation Conference*, pages 927–930, 2009.

[11] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Haichen Shen, Eddie Q Yan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: end-to-end optimization stack for deep learning. *arXiv preprint arXiv:1802.04799*, 2018.

[12] Richard Cole, Vasilis Gkatzelis, and Gagan Goel. Mechanism design for fair division: allocating divisible items without payments. In *Proceedings of the Fourteenth ACM Conference on Electronic Commerce*, pages 251–268, 2013.

[13] Shail Dave, Riyadh Baghdadi, Tony Nowatzki, Sasikanth Avancha, Aviral Shrivastava, and Baoxin Li. Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights. *Proceedings of the IEEE*, pages 1706–1752, 2021.

[14] ONNX Runtime developers. Onnx runtime. https://onnxruntime.ai/, 2021.

[15] Li Du and Yuan Du. Hardware accelerator design for machine learning. In *Machine Learning-Advanced Techniques and Emerging Applications*. IntechOpen, 2017.

[16] Allan Feldman and Alan Kirman. Fairness and envy. *The American Economic Review*, pages 995–1005, 1974.

[17] Matthias Feurer and Frank Hutter. Hyperparameter optimization. *Automated Machine Mearning: Methods, Systems, Challenges*, pages 3–33, 2019.

[18] Giannis Fikioris, Rachit Agarwal, and Éva Tardos. Incentives in dominant resource fair allocation under dynamic demands. *arXiv preprint arXiv:2109.12401*, 2021.

[19] Duncan Karl Foley. *Resource allocation and the public sector*. Yale University, 1966.

[20] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, pages 119–139, 1997.

[21] Wei Gao, Qinghao Hu, Zhisheng Ye, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. Deep learning workload scheduling in GPU datacenters: Taxonomy, challenges and vision, 2022.

[22] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *Proceedings of the 58th Annual Design Automation Conference (DAC)*, 2021.

[23] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[24] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020.

[25] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, pages 1291–1307, 2012.

[26] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 485–500. USENIX Association, 2019.

[27] Suyog Gupta and Berkin Akin. Accelerator-aware neural network design using automl. *arXiv preprint arXiv:2003.02838*, 2020.

[28] Hamed Hamzeh, Sofia Meacham, Botond Virginas, Kashaf Khan, and Keith Phalp. MLF-DRS: A multi-level fair resource allocation algorithm in heterogeneous cloud computing systems. In *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, pages 316–321. IEEE, 2019.

[29] Dorothea K Herreiner and Clemens D Puppe. Envy freeness in experimental fair division problems. *Theory and Decision*, pages 65–100, 2009.

[30] Randall G Holcombe. Absence of envy does not imply fairness. *Southern Economic Journal*, pages 797–802, 1997.

[31] Eunji Hwang, Jik-Soo Kim, and Young-ri Choi. Achieving fairness-aware two-level scheduling for heterogeneous distributed systems. *IEEE Transactions on Services Computing*, pages 639–653, 2018.

[32] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 968–981. IEEE, 2020.

[33] Miia Jansson, Janne Liisanantti, Tero Ala-Kokko, and Jarmo Reponen. The negative impact of interface design, customizability, inefficiency, malfunctions, and information retrieval on user experience: A national usability survey of icu clinical information systems in finland. *International Journal of Medical Informatics*, page 104680, 2022.

[34] Myeongjae Jeon, Shivaram Venkataraman, Junjie Qian, Amar Phanishayee, Wencong Xiao, and Fan Yang. Multi-tenant GPU clusters for deep learning workloads: Analysis and implications. *Technical report, Microsoft Research*, 2018.

[35] Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. Multiresource allocation: Fairness–efficiency tradeoffs in a unifying framework. *IEEE/ACM Transactions on Networking*, pages 1785–1798, 2013.

[36] Jalal Khamse-Ashari, Ioannis Lambadaris, George Kesidis, Bhuvan Urgaonkar, and Yiqiang Zhao. An efficient and fair multi-resource allocation mechanism for heterogeneous servers. *IEEE Transactions on Parallel and Distributed Systems*, pages 2686–2699, 2018.

[37] Tahseen Khan, Wenhong Tian, Guangyao Zhou, Shashikant Ilager, Mingming Gong, and Rajkumar Buyya. Machine learning (ML)-centric resource management in cloud computing: A review and future directions. *Journal of Network and Computer Applications*, 2022.

[38] Luke Kljucaric and Alan D George. Deep learning inferencing with high-performance hardware accelerators. *ACM Transactions on Intelligent Systems and Technology*, pages 1–25, 2023.

[39] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 1999.

[40] David Koufaty, Dheeraj Reddy, and Scott Hahn. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European conference on Computer systems*, pages 125–138, 2010.

[41] Haidong Lan, Jintao Meng, Christian Hundt, Bertil Schmidt, Minwen Deng, Xiaoning Wang, Weiguo Liu, Yu Qiao, and Shengzhong Feng. Feathercnn: Fast inference computation with TensorGEMM on ARM architectures. *IEEE Transactions on Parallel and Distributed Systems*, pages 580–594, 2019.

[42] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14. IEEE, 2021.

[43] Jiansong Li, Xueying Wang, Xiaobing Chen, Guangli Li, Xiao Dong, Peng Zhao, Xianzhi Yu, Yongxin Yang, Wei Cao, Lei Liu, et al. An application-oblivious memory scheduling system for DNN accelerators. *ACM Transactions on Architecture and Code Optimization (TACO)*, pages 1–26, 2022.

[44] Mingzhen Li, Yi Liu, Xiaoyan Liu, Qingxiao Sun, Xin You, Hailong Yang, Zhongzhi Luan, Lin Gan, Guangwen Yang, and Depei Qian. The deep learning compiler: A comprehensive survey. *IEEE Transactions on Parallel and Distributed Systems*, pages 708–727, 2020.

[45] Weidong Li, Xi Liu, Xiaolu Zhang, and Xuejie Zhang. Multi-resource fair allocation with bounded number of tasks in cloud computing systems. In *Theoretical Computer Science: 35th National Conference, NCTCS 2017, Wuhan, China, October 14-15, 2017, Proceedings*, pages 3–17. Springer, 2017.

[46] Xiangru Lian, Binhang Yuan, Xuefeng Zhu, Yulong Wang, Yongjun He, Honghuan Wu, Lei Sun, Haodong Lyu, Chengjun Liu, Xing Dong, et al. Persia: An open, hybrid system scaling deep learning-based recommenders up to 100 trillion parameters. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3288–3298, 2022.

[47] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 789–801, 2021.

[48] Jessica Lindblom, Beatrice Alenljung, and Erik Billing. Evaluating the user experience of human–robot interaction. *Human-Robot Interaction: Evaluation Methods and Their Standardization*, pages 231–256, 2020.

[49] Fang Liu, Guoming Tang, Youhuizi Li, Zhiping Cai, Xingzhou Zhang, and Tongqing Zhou. A survey on edge computing systems and tools. *Proceedings of the IEEE*, pages 1537–1562, 2019.

[50] Haikun Liu and Bingsheng He. Reciprocal resource fairness: Towards cooperative multiple-resource fair sharing in iaas clouds. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 970–981. IEEE, 2014.

[51] Dinh The Luc. Pareto optimality. *Pareto Optimality, Game Theory and Equilibria*, pages 481–515, 2008.

[52] Raju Machupalli, Masum Hossain, and Mrinal Mandal. Review of asic accelerators for deep neural network. *Microprocessors and Microsystems*, page 104441, 2022.

[53] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 289–304. USENIX Association, 2020.

[54] Suchintan Mishra, Arun Kumar Sangaiah, Manmath Narayan Sahoo, and Sambit Bakshi. Pareto-optimal cost optimization for large scale cloud systems using joint allocation of resources. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–19, 2019.

[55] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Analysis and exploitation of dynamic pricing in the public cloud for ml training. In *VLDB DISPA Workshop 2020*, 2020.

[56] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–14, 2018.

[57] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 1–18. USENIX Association, 2021.

[58] Rasmus V Rasmussen and Michael A Trick. Round robin scheduling–a survey. *European Journal of Operational Research*, pages 617–636, 2008.

[59] Gemma Reig, Javier Alonso, and Jordi Guitart. Prediction of job resource requirements for deadline schedulers to manage high-level slas on the cloud. In *2010 Ninth IEEE International Symposium on Network Computing and Applications*, pages 162–167. IEEE, 2010.

[60] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and benchmarking of machine learning accelerators. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9. IEEE, 2019.

[61] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and benchmarking of machine learning accelerators. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, 2019.

[62] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. AI and ML accelerator survey and trends. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–10. IEEE, 2022.

[63] Cong Shen, Jie Xu, and Mihaela van der Schaar. Silence is gold: Strategic interference mitigation using tokens in heterogeneous small cell networks. *IEEE Journal on Selected Areas in Communications*, pages 1097–1111, 2015.

[64] Rakesh Shrestha, Rojeena Bajracharya, Ashutosh Mishra, and Shiho Kim. AI accelerators for cloud and server applications. In *Artificial Intelligence and Hardware Accelerators*, pages 95–125. Springer, 2023.

[65] Ajit Singh, Priyanka Goyal, and Sahil Batra. An optimized round robin scheduling algorithm for cpu scheduling. *International Journal on Computer Science and Engineering*, pages 2383–2385, 2010.

[66] Sharanyan Srikanthan. *Sharing-aware Resource Management for Multicore Systems*. University of Rochester, 2019.

[67] Joseph E Stiglitz. Pareto optimality and competition. *The Journal of Finance*, pages 235–251, 1981.

[68] Shanjiang Tang, BingSheng He, Shuhao Zhang, and Zhaojie Niu. Elastic multi-resource fairness: balancing fairness and efficiency in coupled CPU-GPU architectures. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 875–886. IEEE, 2016.

[69] Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, pages 58–68, 1995.

[70] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). *ACM SIGARCH Computer Architecture News*, pages 213–224, 2012.

[71] Hal R Varian. Equity, envy, and efficiency. 1973.

[72] Wei Wang, Baochun Li, and Ben Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 583–591. IEEE, 2014.

[73] Wei Wang, Ben Liang, and Baochun Li. Multi-resource fair allocation in heterogeneous cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, pages 2822–2835, 2014.

[74] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960. USENIX Association, 2022.

[75] Bingyang Wu, Zili Zhang, Zhihao Bai, Xuanzhe Liu, and Xin Jin. Transparent GPU sharing in container clouds for deep learning workloads. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 69–85, 2023.

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

[77] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. AntMan: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 533–548. USENIX Association, 2020.

[78] Jie Xu and Mihaela Van Der Schaar. Token system design for autonomic wireless relay networks. *IEEE Transactions on Communications*, pages 2924–2935, 2013.

[79] Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A Runtime-Based computational performance predictor for deep neural network training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 503–521. USENIX Association, 2021.

[80] Peifeng Yu and Mosharaf Chowdhury. Fine-grained GPU sharing primitives for deep learning applications. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, pages 98–111, 2020.

[81] Seyed Majid Zahedi, Songchun Fan, and Benjamin C Lee. Managing heterogeneous datacenters with tokens. *ACM Transactions on Architecture and Code Optimization (TACO)*, pages 1–23, 2018.

[82] Seyed Majid Zahedi and Benjamin C Lee. Ref: Resource elasticity fairness with sharing incentives for multiprocessors. *ACM Sigplan Notices*, pages 145–160, 2014.

[83] Seyed Majid Zahedi and Benjamin C Lee. Sharing incentives and fair division for multiprocessors. *IEEE Micro*, pages 92–100, 2015.