applied sciences

Article

Round-Based Mechanism and Job Packing with Model-Similarity-Based Policy for Scheduling DL Training in GPU Cluster

Panissara Thanapol *,†, Kittichai Lavangnananda *,†, Franck Leprévost †, Arnaud Glad †, Julien Schleich † and Pascal Bouvry †

Department of Computer Science, University of Luxembourg, 4365 Luxembourg, Luxembourg; franck.leprevost@uni.lu (F.L.); arnaud.glad@uni.lu (A.G.); julien.schleich@uni.lu (J.S.); pascal.bouvry@uni.lu (P.B.)
* Correspondence: panissara.thanapol@uni.lu (P.T.); kittichai.lav@gmail.com (K.L.)
† These authors contributed equally to this work.

Abstract: Graphics Processing Units (GPUs) are employed for their parallel processing capabilities, which are essential to train deep learning (DL) models with large datasets within a reasonable time. However, the diverse GPU architectures exhibit variability in training performance depending on DL models. Furthermore, factors such as the number of GPUs for distributed training and batch size significantly impact training efficiency. Addressing the variability in training performance and accounting for these influential factors are critical for optimising resource usage. This paper presents a scheduling policy for DL training tasks in a heterogeneous GPU cluster. It builds upon a model-similarity-based scheduling policy by implementing a round-based mechanism and job packing. The round-based mechanism allows the scheduler to adjust its scheduling decisions periodically, whereas job packing optimises GPU utilisation by fitting additional jobs into a GPU that trains a small model. Results show that implementing a round-based mechanism reduces the makespan by approximately 29%, compared to the scenario without it. Additionally, integrating job packing further decreases the makespan by 5%.

Keywords: deep learning; deep learning training; distributed training; GPU cluster; job packing; round-based mechanism; similarity analysis

check for updates

Citation: Thanapol, P.; Lavangnananda, K.; Leprévost, F.; Glad, A.; Schleich, J.; Bouvry, P. Round-Based Mechanism and Job Packing with Model-Similarity-Based Policy for Scheduling DL Training in GPU Cluster. *Appl. Sci.* **2024**, *14*, 2349. https://doi.org/10.3390/app14062349

Academic Editor: Christos Bouras

Received: 6 February 2024
Revised: 5 March 2024
Accepted: 8 March 2024
Published: 11 March 2024

1. Introduction

Deep learning (DL) represents a paradigm within machine learning (ML) wherein iterative learning occurs through neural networks processing input data to acquire the problem-solving ability [1]. Over the past decade, DL has garnered significant success across diverse domains, including but not limited to image processing and natural language processing [2,3]. Despite these achievements, it is noteworthy that training DL models is a time-consuming and resource-intensive task [4]. In order to reduce the consumption of time, Graphics Processing Units (GPUs) for parallel processing have emerged as a viable alternative to accelerate the training process [5]. Furthermore, the adoption of parallel training across multiple GPUs proves to be a practical strategy for reducing the overall training time. Intensive demand of resources is a much harder issue to mitigate, as it depends largely on the complexity of DL models (i.e., architectures) and amount of data used.

Nowadays, a multitude of GPU architectures present themselves as viable choices for training DL models. Nevertheless, determining GPU performance, especially in the context of DL, cannot solely rely on published performance data, as prevailing DL models are often trained for benchmarking purposes. It is essential to recognise that the most recently released GPU architecture may not necessarily constitute the optimal choice for certain models [6]. The training performance is contingent upon factors such as GPU architecture, the number of GPUs used, hyperparameters (such as the batch size, which must be tuned according to the available GPU memory), and the specific characteristics inherent to the DL model (such as the number of layers or the activation function) under consideration [7]. It then becomes clear that selecting the most suitable GPU architecture for a specific model significantly expedites the training process [8].

Training DL models can be conducted within a GPU cluster, potentially featuring heterogeneous GPU architectures. This diversity in GPU architectures provides a range of options for DL users. It adds complexity to the training, as these options mainly rely on the users' experience. Even for experienced users, choosing GPU architectures is often a matter of trial and error.

Consequently, in a heterogeneous GPU cluster, an effective scheduler must both manage resources within a multi-tenant environment and consider the variable training performance inherent to DL models. Therefore, this paper focuses on designing scheduling policies that take these aspects into account to minimise the time required to complete a set of jobs.

With insights into the variables affecting training performance, cutting-edge scheduling techniques can enhance scheduling policies for DL training tasks, especially within a heterogeneous GPU cluster. Firstly, a round-based mechanism enables a scheduler to periodically re-schedule, addressing potential sub-optimal allocations [8]. Secondly, job packing emerges as a valuable strategy for augmenting cluster utilisation. It is particularly effective when dealing with small model training in isolation, as this scenario typically results in the underutilisation of a GPU. Simultaneously, job packing reduces job waiting time [9]. Those approaches provide good performances and can be combined when establishing a scheduling policy. This paper presents a new approach combining the works of Deepak et al. [8] and Gandiva [9] as well as building upon our earlier research [7] to extend the contributions made in the previous study.

The key contributions of the paper are as follows:

- **Implementation of a round-based mechanism:** We implement a round-based mechanism to take advantage of re-scheduling.
- **Integration of job packing:** We incorporate the concept of job packing to enhance cluster utilisation and reduce waiting time.
- **Optimising job throughput:** This work emphasises the optimisation of job throughput or the amount of data that a job processes within a given time, in accordance with the approach in [8], instead of focusing on training time. This approach allows for a meaningful comparison with state-of-the-art scheduling policies.
- **Dynamic job allocations:** We propose job allocations that can be dynamically adjusted based on specific objective functions, optimising training performance based on the GPU architecture choice, the number of GPUs, and batch size for a given DL model.

The organisation of this paper is as follows. A literature review is presented in Section 2. Section 3 starts with problem formulation, and then the design of our proposed scheduling policy is discussed in Section 4. Section 5 describes the parameters to evaluate this study. This is followed by Section 6, where the results and the discussion of this study are presented. This paper is concluded in Section 7, where the contributions are summarised, and possible future works are suggested.

2. Literature Review

In recent years, numerous studies have proposed scheduling policies for DL training tasks within GPU clusters that leverage ML approaches. For instance, Optimus [10] predicts the number of epochs needed for model training and utilises this information to estimate the total training time on allocated resources. Similarly, the approach presented in [11] develops predictors for the training time of specific network components, such as fully connected and convolutional layers. In this case, the overall training time is the sum of individual times for each component. Furthermore, the study outlined in [12] introduces a method for predicting resource consumption based on GPU selection across various DL models, enabling the estimation of training time beforehand. Xonar [13] also takes into account resource consumption by profiling job memory requirements to ensure that a job can receive sufficient memory to execute, thereby avoiding the out-of-memory (OoM) problem. On the other hand, works in [14–16] employ Reinforcement Learning (RL) to formulate scheduling policies. It is noteworthy, however, that these ML-related scheduling policies necessitate a substantial volume of data for training.

While the variability of training performance in a heterogeneous GPU cluster has been explored in prior research, the extent of such investigations remains limited. Notably, Gavel [8] and Habitat [6] select GPU architectures for jobs based on the DL model but do not encompass other crucial factors such as the number of GPUs used for distributed training and hyperparameters (e.g., batch size). More importantly, the batch size is emphasised as one of the necessary consideration variables of a computation-related system [17].

As the DL training process is iterative, a scheduler can derive benefits from the ability to suspend and resume at specific points (i.e., checkpointing) and reschedule the job with better resources when they become available. Various scheduling policies have been devised on this premise, such as Gandiva [9], DL2 [18], and Optimus [10]. These policies allocate resources to jobs and assess training performance dynamically to achieve satisfactory results, but, ultimately, this approach is still far from optimal.

Gandiva [9] and Gavel [8] reveal that a small model often underutilises a GPU. To address this issue, they have introduced the concept of job packing, enabling the simultaneous training of multiple models on a single GPU. However, these studies highlight that concurrent jobs can potentially interfere with each other, adversely affecting training performance. Furthermore, the extent of interference depends on the DL models themselves [19,20]. In the pursuit of identifying suitable job combinations, Gandiva employs a trial-and-error approach, while Gavel establishes a threshold for the difference between isolated training and packing decisions. In both cases, there are limitations. Gandiva spends training time searching for better solutions, and Gavel uses a constant threshold that is set empirically.

3. Problem Formulation

The scheduling policy in this work can be seen as an optimisation problem. In this section, we formulate the optimisation problem we aim to solve. We then provide an overview of the DL training process and define our optimised variable. Additionally, insights into the DL training performance relevant to our optimisation problem are illustrated.

3.1. Optimisation Problem

The optimisation problem of this work and its relevant elements are defined as follows:

Objective function. This study addresses an optimisation problem for identifying the optimal set of allocation variables to maximise job throughput. In the context of job packing, the objective is to maximise the total throughput of a set of jobs. The definition of job throughput is given in Section 3.2 for a more comprehensive understanding.

Decision variables. In this study, we consider the following allocation variables to optimise throughput: GPU architecture, the number of GPUs used for training, and the batch size. Each of these variables plays a distinct role in shaping the objective function. We present an analysis of their impact on the objective function in Section 3.3.

Constraints. The optimisation problem is subject to the following constraints.

- The allocation of jobs is made in rounds (i.e., the allocation is carried out at a set frequency, every t time unit).
- The allocations made in each round must ensure they do not oversubscribe resource availability.
- After each round, the allocation of jobs can change.
- The packed job must not exceed a specific GPU memory.
- Job allocation is made in sequence based on job arrival time. However, in the case of job packing, where GPU memory is a consideration, the priority to arrival time is relaxed.

3.2. Throughput of Training DL Model over GPU

In DL, the training process notably relies on the Stochastic Gradient Descent (SGD) optimisation algorithm. Its main objective is to optimise (or train) the parameters of a model in order to minimise prediction error by performing the following steps:

- predict results based on the current state of the model and the training dataset;
- calculate the prediction error;
- update the model parameter.

The number of samples used in the aforementioned steps is referred to as batch size, i.e., the number of samples of the training dataset which are run through the model before updating its parameters based on the quality of the prediction.

One epoch means that each sample in the training dataset was used to update the model parameters. An epoch is usually composed of several batches, and the number of batches that composes one epoch can be referred to as the number of iterations; see Figure 1. For example, the CIFAR-10 dataset [21] is composed of over 60,000 images, and thus, a batch size of 32 would require 1875 iterations to complete one epoch. Finally, the training of a DL model usually requires multiple epochs to reach a desired level of quality.

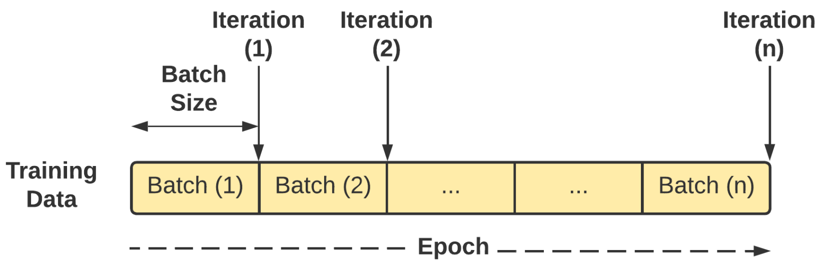


Figure 1. Illustration of batch size, iteration, and epoch.

There are multiple definitions of throughput; we consider the one defined in [8]. It refers to the number of iterations a model can train per second, which is optimised in the problem defined in Section 3. Normally, the framework for DL logs the training time per epoch while training. The throughput can be calculated from the training time per epoch, as seen below:

$$iterations = \frac{total\_images}{batch\_size} \tag{1}$$

$$throughput = \frac{iterations}{time\_per\_epoch} \tag{2}$$

where *total_images* is the total of images in the dataset. *batch_size* is the batch size configured for each job. *time_per_epoch* is the training time per epoch recorded when training a model in a different GPU architecture.

3.3. Exploring the Influence of Allocation Variables on the Objective Function

We present here the allocation variables described in the problem formulation to explore their impact on throughput.

The relationship of batch size to the objective function. Small batch sizes (e.g., 32) tend to prolong the training duration, achieve a higher throughput, and enhance the model performance. This is due to the fact that, as batch size decreases, the number of iterations in an epoch, and thus, the number of model parameter updates, increases, as illustrated in Figure 2. When considering the objective function, reducing the batch size can maximise throughput and model performance, while increasing the batch size can minimise the training time.

Influence of distributed training on job throughput. Referring to Figure 3, the throughput for multiple GPUs while training a VGG16 model on the RTX2080Ti GPU architecture declines as the number of GPUs used for the data parallelism approach in distributed training increases. An exception is observed for a batch size of 512, where using multiple GPUs in training results in slightly higher throughput. As a result, reducing the batch size while training on a single GPU can yield a significantly increased throughput compared to distributed training across multiple GPUs. This is due to the communication-intensive nature of distributed training in the context of the data parallelism approach, particularly when dealing with a large batch size, as highlighted in [22].

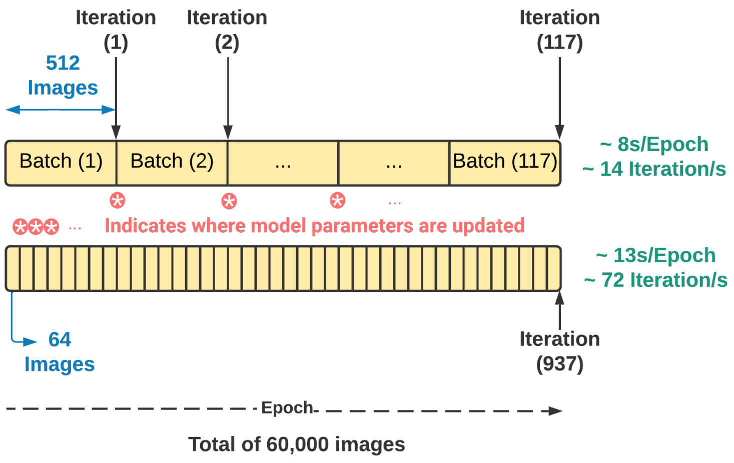


Figure 2. The frequency of model parameter updated on different batch size.

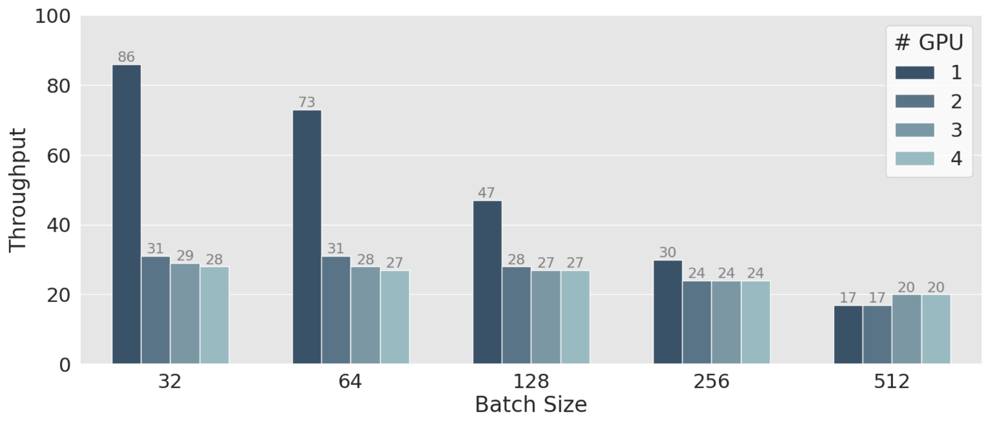


Figure 3. Throughput of various batch sizes of a VGG16 model on several numbers of Nvidia RTX2080Ti GPU architecture.

A decline in job throughput when integrating job packing. When employing job packing for training multiple models on a GPU, the throughput exhibits variations depending on the combination of models. Figure 4 shows the throughput of sharing an Nvidia A100 GPU architecture for concurrent training of a VGG16 model with other models. The batch size of both models is configured as 32. The red line represents the throughput of an isolated training of a VGG16 model. A comparison between isolated training and job packing reveals a decrease in the throughput of the VGG16 model when packed with other models, and the extent of this decrease varies. For instance, the throughput of a VGG16 model packed with another VGG16 model is less than when packed with a ResNet101 model. Even though the throughput of job packing is lower than an isolated training, job packing can reduce queuing delay.

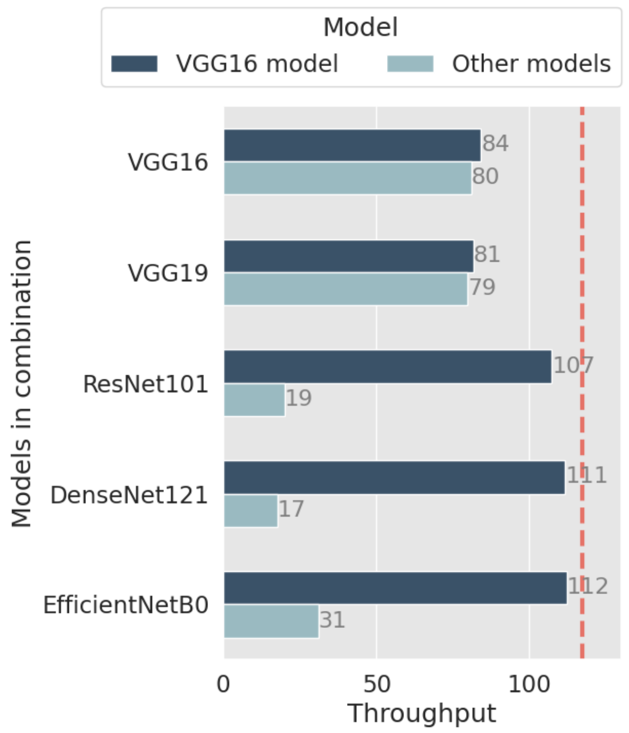


Figure 4. Throughput of sharing an Nvidia A100 GPU architecture for a VGG16 model and other models. The red line represents the baseline throughput for VGG16 training alone.

4. Scheduling Policy

We build upon previous works described in [7]. We develop the model-similarity-based scheduling policy by implementing a round-based mechanism and job packing. For ease of description, the important notations and their definitions used throughout this paper are listed in Table 1.

Table 1. Notations and their definitions.

Notations	Definitions
j	A current job
J	Active jobs in the queue
c_i	Model configurations, $i = 1, \ldots, n$
SJ	Jobs that are scheduled in current round
FJ	A set of jobs recorded in the system
fj	A specific recorded job in the system
G	GPU architectures in the cluster
n_gpu	Number of total GPU availability
g	Specific GPU architecture
ng	Number of a specific GPU
m	The closest reference to the given model j
S	Results of computing similarity between a job j and jobs FJ
sch_m	Suggestion for scheduling a job j based on the closest reference to the given model
jp_m	Potential job combinations based on the closest reference to the given model
T	Total processing time of jobs J
arr	Arrival time
st	Start execution time
wl	Workload of GPU architecture

4.1. The Model-Similarity-Based Scheduling Policy

The model-similarity-based scheduling policy employs a similarity measurement approach. This measurement is computed by comparing a job against reference jobs recorded in a database. Numerous approaches exist to measure the similarity between objects, such as Euclidean distance [23], Manhattan distance [24], and cosine similarity [25]. Specifically, cosine similarity is applied to measure the similarity of the model in this context due to its suitability for handling multi-dimensional data. The model similarity is based on DL characteristics, such as the number of layers and model parameters.

Each job is represented by a set of model characteristics. Let a job j be a tuple $(c_1, c_2, \ldots, c_n)$ of the model characteristics. The current job j is compared to other jobs $fj \in FJ$. The similarity of (j, fj) is defined as the similarity of two objects in a multidimensional space. It is determined by the following:

$$\begin{aligned} similarity(j,fj) = cos(\theta) &= \frac{j \cdot fj}{\|j\| \|fj\|} \\ &= \frac{\sum_{i=1}^{n} f_i \cdot fj_i}{\sqrt{\sum_{i=1}^{n} f_i^2} \sqrt{\sum_{i=1}^{n} fj_i^2}} \end{aligned} \tag{3}$$

where fj denotes a reference job recorded in the system that is currently compared to the current job.

The procedure of computing the similarity between a job j and a set of reference jobs in the database FJ is described in Algorithm 1.

Algorithm 1 Algorithm of computing model similarity.

```
1: $j \leftarrow (c_1, c_2, \ldots, c_n)$
2: for each $fj \in FJ$ do
3:     $S_{(j,fj)} \leftarrow similarity(j, fj)$
4: end for
```

After discovering the closest reference to the given model, a scheduler has the training information of the reference model on several GPU architectures. This information includes GPU architecture, the corresponding number of GPUs used in training, and the batch size. They are then organised in descending order of throughput and supplied to the scheduler to make a scheduling decision based on current cluster availability.

4.2. Implementation of a Round-Based Mechanism

Once a job is submitted to a GPU cluster, its position in the queue is based on its arrival time. Our scheduling policy periodically assigns resources to jobs within the queue, comprising both unfinished jobs from the prior round (i.e., the job is necessarily interrupted and then resumed) and newly submitted jobs.

In every scheduling round, the scheduler sequentially allocates resources to jobs in the queue. It selects the GPU architecture and other allocation variables from a varied set of options for each specific job. Subsequently, resources are assigned to the next job in the queue until no resources remain available. In subsequent rounds, the allocation of jobs can change if a better or more optimal option is available.

Given the results from the model-similarity-based scheduling policy, the scheduler allocates resources to a specific job, aiming to maximise the throughput of each job based on current cluster availability. The overall scheduling is described in Algorithm 2.

However, training a DL model with a synchronous approach in preemptive scheduling can incur overhead when the training is interrupted [26,27]. This overhead includes saving and loading the checkpoint. Saving the checkpoint involves storing the training results from the start of training until the interruption. Loading the checkpoint entails retrieving the previous training results from the checkpoint and resuming training from that point.

Algorithm 2 Algorithm of a round-based mechanism with model-similarity-based scheduling policy.

```
1:  $J \leftarrow$ all active jobs in the queue
2:  $n\_gpu \leftarrow$ the number of total GPU availability
3:  while $n\_gpu > 0$ and $J$ not empty do
4:      $j \leftarrow$ a current job in the queue
5:      $m \leftarrow$ the closest reference to the given model of job $j$
6:      $sch_m \leftarrow$ the suggestions of GPU and its corresponding number
7:      for $(g, ng)$ in $sch_m$ do
8:          $(g, ng) \leftarrow$ the specific GPU and its corresponding number
9:          if $ng \leq n\_gpu_g$ then
10:             Schedule $j$ on $(g, ng)$
11:             $n\_gpu_g \leftarrow n\_gpu_g - ng$
12:             Delete $j$ from queue
13:             break
14:         end if
15:     end for
16: end while
```

The overhead varies depending on several factors, including communication overhead and the characteristics of the DL model. Concerning communication overhead, the overhead occurs when the saved checkpoint is loaded across compute nodes to resume training. On the other hand, overheads associated with the DL model involve factors such as the size of the neural network.

Previous works have highlighted the significance of reducing these overheads to ensure the efficient implementation of a round-based mechanism. The work in [9] investigates the overhead of saving and loading checkpoints on distributed training. It concludes that this process typically takes approximately a few seconds each time and remains stable even as the number of GPUs used in training increases or exceeds the compute node. Additionally, the work in [8] demonstrates that overhead can be reduced by retaining the allocation from the previous round and allocating it to a job in the current round whenever possible. To implement our scheduling policy with a round-based mechanism, we use the framework provided by the work in [8].

4.3. Integration of Job Packing

Job packing improves the utilisation of a GPU allocated to train a small model by efficiently using the remaining resources to accommodate additional jobs. However, it is crucial to ensure that the memory requirement of job combinations does not exceed the GPU memory. The memory required for each job is determined by the model size and batch size. Therefore, we propose to automatically override the user-specified batch size to overcome the limitation of GPU memory and train both models simultaneously on the same GPU.

In addition, job packing helps to alleviate queuing delays by enabling the simultaneous training of multiple models. However, it might introduce some interference that leads to a reduction in job throughput. Job packing is, hence, considered when the cluster load is high (i.e., still having jobs in the queue awaiting resources). In this work, scheduling decisions are initially made without job packing, and if there are more jobs in the queue, the scheduler seeks opportunities to implement job packing.

Previous work indicates that combining at most two jobs is the most effective in terms of throughput [8]. More importantly, the throughput of job combinations varies depending on the chosen models and their batch size. As a consequence, the model-similarity-based scheduling policy is applied to identify the optimal job combinations that maximise job throughput.

Given a list of scheduling decisions for jobs made in each round, a job running alone in a single GPU can be packed with another job in the queue if the GPU memory allows it.

We compute the similarity measure for this job and find the closest reference model in the database. From this, we compute the list of combined throughput between the reference model and all the pending jobs. The scheduler then selects job combinations based on two criteria: minimising a reduction in throughput for the original job and maximising the combined throughput of job combination. Due to the memory constraint, the constraint on the arrival time of jobs in the queue is relaxed, which means any jobs can be chosen to pack. The procedure of job packing is detailed in Algorithm 3.

Algorithm 3 Algorithm of job packing.

```
1:  $SJ \leftarrow$ scheduled jobs in the current round
2:  while $J$ not empty do
3:      for $sj$ in $SJ$ do
4:          if $sj$ is trained alone in a GPU then
5:              $jp_m \leftarrow$ potential jobs in the queue to pack with $sj$
6:              $(sj, j \in jp_m) \leftarrow$ combinations that $max(throughput_{sj})$ and
                        $max(sum(throughput_{(sj,j)}))$
7:              Packing $(sj, j)$
8:              Delete $j$ from queue
9:          end if
10:     end for
11: end while
```

5. Experimentation Setup

This section provides the experimentation setup to evaluate our scheduling policy. It includes jobs, the considered GPU architectures, as well as descriptions of the experiments. Finally, we define the metrics to evaluate the performance of the scheduling policy.

5.1. Jobs and Cluster Used in the Evaluation

Jobs. There are several DL model architectures, such as Recurrent Neural Networks (RNNs) [28] and Convolutional Neural Networks (CNNs) [29]. In our evaluation, we adopt CNN as the DL model, trained with the CIFAR-10 dataset. This selection is based on the extensive popularity of CNNs in image processing and the wide accessibility of the CIFAR-10 dataset. Our evaluation contains a diverse set of 21 CNN models, each varying in characteristics such as model size, number of parameters, and the number of layers. The selection of models used in our evaluation is based on the availability of pre-built models in the Keras framework [30], covering a range from small to large models. The batch size for each model is configured within the range of 32 to 512 (maximum at 512 due to GPU memory constraint).

Jobs are defined as a 4-tuple: DL model name, batch size, GPU architecture, and GPU count, as detailed in a set in Table 2. An example of a job is the training of a VGG16 model with a batch size of 32 on an A100 GPU architecture with 1 GPU. In this study, it gives us a total of 1680 possible combinations.

The number of jobs is configured in four settings: 25, 50, 100, and 200. They are uniformly sampled from the job table, shown in Table 2. The jobs are submitted at a regular pace over a given period of time. The selected scenarios include a 0–15 min span and a 0–7 min span, with all jobs arriving simultaneously at time 0.

Cluster. This study considers a heterogeneous GPU cluster comprising six different GPU architectures: Nvidia A100, A40, RTX2080Ti, RTX1080Ti, K40M, and T4. The machine of each GPU architecture contains multiple GPUs, with its count indicated in Table 2. Due to the accessibility constraints of the GPU cluster, the experiments in this work are conducted through simulation. In our study, the simulation specifically focuses on two key aspects: the performance of each GPU architecture and the memory specifications of the GPUs. The reference training time of all jobs on the six different GPU architectures was recorded on a real cluster (Grid'5000 [31]) and reused in the following simulations.

Table 2. Jobs and GPU architectures used in evaluation.

Components	Elements	
DL models	{1..21} CNN models	
	Their number of layers and model parameters are described as follows:	
VGG [32]	VGG16:	16 layers with 138.4 M parameters
	VGG19:	19 layers with 143.7 M parameters
ResNet [33]	ResNet50:	107 layers with 25.6 M parameters
	ResNet50V2:	103 layers with 25.6 M parameters
	ResNet101:	209 layers with 44.7 M parameters
	ResNet101V2:	205 layers with 44.7 M parameters
	ResNet152:	311 layers with 60.4 M parameters
	ResNet152V2:	307 layers with 60.4 M parameters
MobileNet [34,35]	MobileNet:	55 layers with 4.3 M parameters
	MobileNetV2:	105 layers with 3.5 M parameters
DenseNet [36]	DenseNet121:	242 layers with 8.1 M parameters
	DenseNet169:	338 layers with 14.3 M parameters
	DenseNet201:	402 layers with 20.2 M parameters
EfficientNet [37]	EfficientNetB0:	132 layers with 5.3 M parameters
	EfficientNetB1:	186 layers with 7.9 M parameters
	EfficientNetB2:	186 layers with 9.2 M parameters
	EfficientNetB3:	210 layers with 12.3 M parameters
	EfficientNetB4:	258 layers with 19.5 M parameters
	EfficientNetB5:	312 layers with 30.6 M parameters
	EfficientNetB6:	360 layers with 43.3 M parameters
	EfficientNetB7:	438 layers with 66.7 M parameters
Batch size	$\{32, 64, 128, 256, 512\}$	
GPU architecture and GPU count	Nvidia A100 $\{1, 2\}$	
	Nvidia A40 $\{1, 2\}$	
	Nvidia RTX1080Ti $\{1, 2\}$	
	Nvidia K40M $\{1, 2\}$	
	Nvidia RTX2080Ti $\{1, 2, 3, 4\}$	
	Nvidia T4 $\{1, 2, 3, 4\}$	
Total	1680 jobs	

5.2. Experiments

Scheduling decision. The decision variables influencing the training performance, which are GPU architecture, the number of GPUs used in training, and batch size, are defined in Section 3. The adjustment of the batch size is an optional service within a GPU cluster to improve cluster efficiency. We will conduct the experiments by varying these three parameters. We divide the experiments as follows.

1. **Making a decision for GPU architecture.** The scheduler selects the GPU architecture for a given job to maximise its throughput. We assume that users provide information about the number of GPUs used in training and the batch size.
2. **Making a decision for GPU architecture and number of GPUs used in training.** The scheduler selects GPU architecture and its number in training for a given job to maximise its throughput. In this case, we assume that users configure the batch size.
3. **Making a decision for GPU architecture, number of GPUs used in training, and adjustment of batch size.** In practice, batch size is a user choice, and the scheduler does not modify the user-identified batch size. However, suggesting a batch size along with resource allocation can be an optional service, potentially achieving higher throughput based on the allocated resources. In this experiment, the scheduler selects GPU architecture and its number in training and batch size for a given job to maximise its throughput. The results can be compared with the experiments without changing batch size to evaluate the improvement.

Time per iteration. The scheduling decisions are re-evaluated periodically with a round duration of 3, 6, 12, and 30 min. The experiments are conducted under two conditions: a scheduling decision that can be changed in the next round and, as a control case, a scheduling decision without any changes. We repeat the experiment ten times for each possible parameter combination and average the results.

5.3. Evaluation Metrics

The following are the metrics used to evaluate the efficacy of scheduling policy.

Makespan. Makespan is a simple metric for scheduling problems representing the completion time of the last job [38]. In this work, there are six GPU architectures in the cluster. Thus, makespan is the time for the last finished job among all GPU architectures. The makespan MK is calculated by:

$$MK = \max(T_g \in G) \tag{4}$$

where T_g denotes the total processing time of GPU architecture g.

Average job completion time. Job completion time is the time from submission to completion. Unlike makespan, the average job completion time can give information on a reduction in training time without considering waiting time. The average job completion time JCT is calculated by:

$$JCT = \frac{1}{J} \sum_{j \in J} t_j \tag{5}$$

where t_j denotes the training time of job j.

Average job waiting time. Job waiting time refers to a delay in executing a job. It is the difference between the start execution time and the arrival time. The average waiting time WT is calculated by:

$$WT = \frac{1}{J} \sum_{j \in J} st_j - arr_j \tag{6}$$

where st_j is the start execution time, and arr_j is the arrival time of job j.

Cluster utilisation. GPU utilisation is the percentage of GPU processing over a particular time. In this work, the cluster is composed of six different GPU architectures. Therefore, cluster utilisation is an average percentage of all GPU architecture processing times. The cluster utilisation $cluster\_util$ is calculated by:

$$cluster\_util = \frac{\frac{1}{n\_gpu} \sum_{wl_g \in G}}{T} * 100 \tag{7}$$

where wl_g is the workload of a specific GPU architecture in the cluster.

6. Results and Discussion

This section presents the results and compares them with the state-of-the-art scheduling policies.

6.1. Comparison among Our Experiments

For conciseness, the results of the different experiments presented are labelled with abbreviations. The experiments are over three allocation variables: GPU architecture, number of GPUs used in training, and batch size. They are abbreviated as *gpu*, *ngpu*, and *bs*, respectively. The experiments described in Section 5.2 are represented by *similarity_gpu* (Experiment 1), *similarity_gpu_ngpu* (Experiment 2), and *similarity_gpu_ngpu_bs* (Experiment 3). Furthermore, the experiments in which the scheduling decisions can be changed for the next round are denoted as *cd*, and the experiments integrating job packing are denoted as *jp*.

Comparison of makespan on three different aspects. Figure 5 illustrates the makespan comparison across three key aspects of our experiments. These aspects include varying time per iteration (columns), three job density scenarios (ranges of arrival time in rows), and the increasing number of jobs (x-axis). The makespan generally rises due to two factors: an increased number of jobs necessitating more time for completion and the gradual delay of job arrivals.

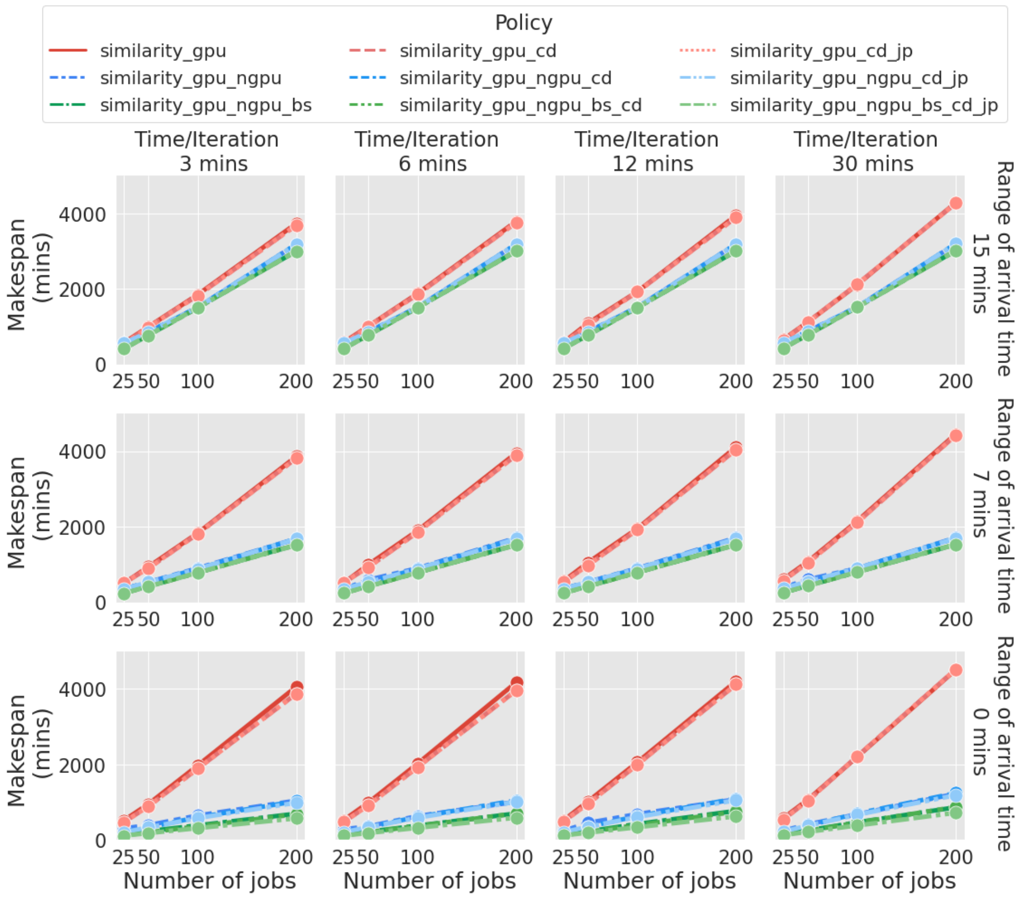
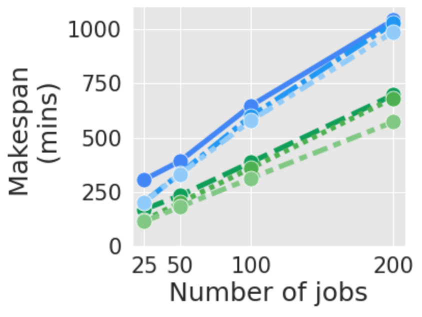


Figure 5. Comparison of makespan among experiments on three different aspects.

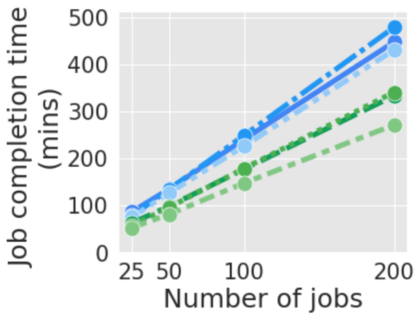
In the comparison of job density scenarios, the makespan slope varies across arrival time ranges. The makespan of *similarity_gpu* remains consistently high while *similarity_gpu_ngpu* and *similarity_gpu_ngpu_bs* show comparable makespans, with a noticeable difference observed when all jobs arrive simultaneously (range of arrival time of 0 min). In this scenario, the makespan of *similarity_gpu_ngpu_bs* is slightly lower than that of *similarity_gpu_ngpu*.

In considering time per iteration for rescheduling, one can observe an increase in both makespan and time per iteration as the interval between rescheduling grows. This is due to potential GPU architecture idleness when a job finishes during a round, and rescheduling does not occur until the end of that round. In our study, rescheduling every three minutes yields the lowest makespan, although there is a slight gap between rescheduling every three and six minutes.

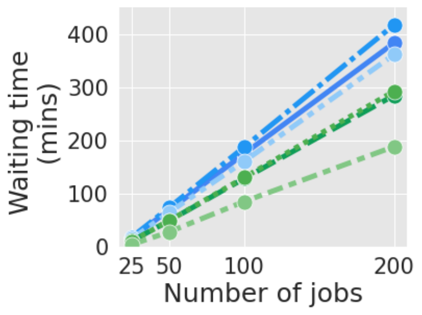
Improvement of implementing a round-based mechanism. Figure 6a–d compare various evaluation metrics between two experiments (*similarity_gpu_ngpu* and *similarity_gpu_ngpu_bs*) in the scenario that all jobs arrive simultaneously, with rescheduling every three minutes. The results of the *similarity_gpu* experiment are excluded from this figure, as they do not show any significant improvement. The lack of improvement arises from the scheduler focusing exclusively on selecting the appropriate GPU architecture for jobs while overlooking the impact of the number of GPUs used, which also playing a pivotal role in training performance.



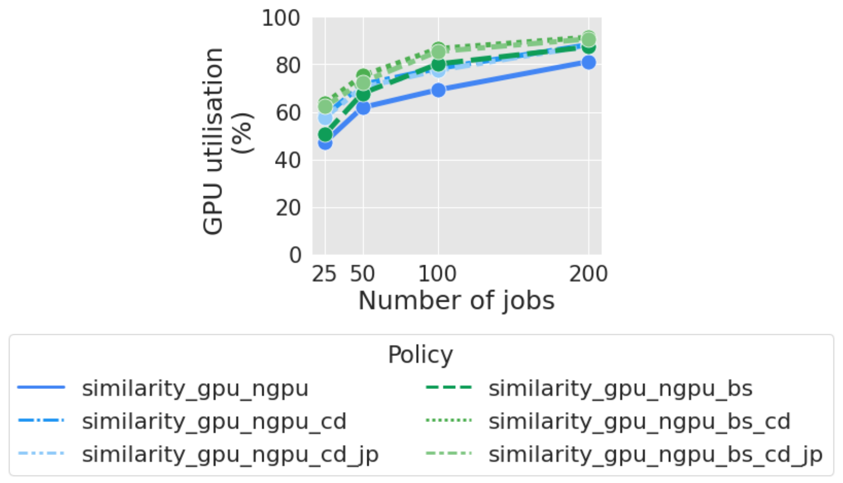
(**a**) Makespan



(**b**) Job completion time



(**c**) Waiting time



(**d**) GPU utilisation

Figure 6. Comparison of several evaluation metrics among experiments in the scenario that all jobs arrive simultaneously with rescheduling every three minutes. The blue represents the experiment of *similarity_gpu_ngpu*. The green represents the experiment of *similarity_gpu_ngpu_bs*.

Incorporating the model-similarity-based scheduling policy with a round-based mechanism, as illustrated in experiments *similarity_gpu_ngpu_cd* and *similarity_gpu_ngpu_bs_cd*, results in a reduction in makespan and an increase in GPU utilisation compared to experiments without the allowance to change the scheduling decision. Furthermore, the average job completion time decreases, particularly with a certain number of jobs (25 and 50 jobs, in our case). Despite these advantages, there is a slight delay in the queue.

The experiments of *similarity_gpu_ngpu_cd* and *similarity_gpu_ngpu_bs_cd* with 25 jobs show a reduction in makespan by approximately 33% and 29%, respectively, compared to their counterparts without the round-based mechanism that allows changing the scheduling decision in each round (i.e., without the *cd* label). The improvement in GPU utilisation is approximately 24% and 26%, respectively, compared to their counterparts without the round-based mechanism. Also, the average job completion time also decreases by approximately 10% and 9%, respectively, compared to their counterparts without the round-based mechanism. The extent of improvement varies as the number of jobs increases.

Improvement of integrating job packing. Figure 6 shows an improvement along all considered metrics when using job packing (light blue and light green dotted lines). The most significant gains come when applying job packing along with adjusting the batch size (*similarity_gpu_ngpu_bs_cd_jp*).

Figure 6a–d show that the experiment of *similarity_gpu_ngpu_bs_cd_jp* outperforms others. As the number of jobs increases, its performance significantly surpasses the approaches without job packing. At the high load of 200 jobs, the *similarity_gpu_ngpu_bs_cd_jp* experiment decreases the makespan by approximately 16% compared to its counterpart with a round-based mechanism but without job packing. It decreases the average job completion time by approximately 20% and the average waiting time by approximately 35%.

6.2. Comparison with the State of the Art

We compare our scheduling policy with the state-of-the-art scheduling policies stated in Table 3.

Differences in scheduling decision making between our work and the state of the art. We should consider the similarities and differences in scheduling decision making between our work and the state of the art.

First, their scheduling decisions can be changed in the subsequent round. Their results can thus be compared with ours under the same conditions, which is the experiments with the *cd* label.

Second, in a real cluster, the training performance remains unknown until a job initiates running for a few epochs, thus preventing the scheduler from making the optimal decision. In this work, we employ the model-similarity-based scheduling policy, which gives us an estimate to overcome this limitation. However, the previous works to which we compare provide the actual throughput recorded in advance to the scheduler, enabling the scheduler to make decisions based on predetermined information. Acting on perfect data that should not be known is not a realistic hypothesis and might skew the results in their favour.

Third, their optimisation focuses on the throughput of a set of jobs. This implies that any jobs fitting the available resources can run, irrespective of their arrival time. This approach may impact job waiting times, as jobs can commence training and then be paused later. In contrast, our optimisation prioritises job arrival time to align with a typical real cluster.

Comparison of several evaluation metrics. Figure 7 compares the makespan of our experiments to the state of the art in three scenarios of job density (range of arrival time 0–15, 0–7 minutes, and all jobs arrive simultaneously), with rescheduling every 3 min. Our work demonstrates superior performance, outperforming the state of the art in makespan across all scenarios. Remarkably, our work exhibits a significant decrease in makespan, particularly in scenarios where job arrivals span 0–7 min and when all jobs arrive simultaneously.

Figure 8a–c compare job completion time, waiting time, and GPU utilisation of our experiments to the state of the art. Our work outperforms the state of the art across all evaluation metrics. A substantial improvement is observed in job completion time and GPU utilisation, while waiting time exhibits a modest improvement.

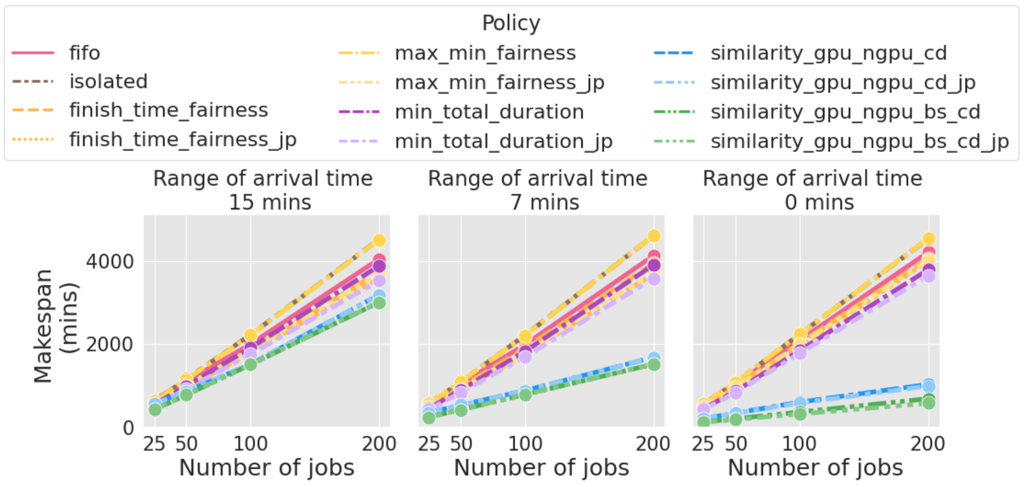
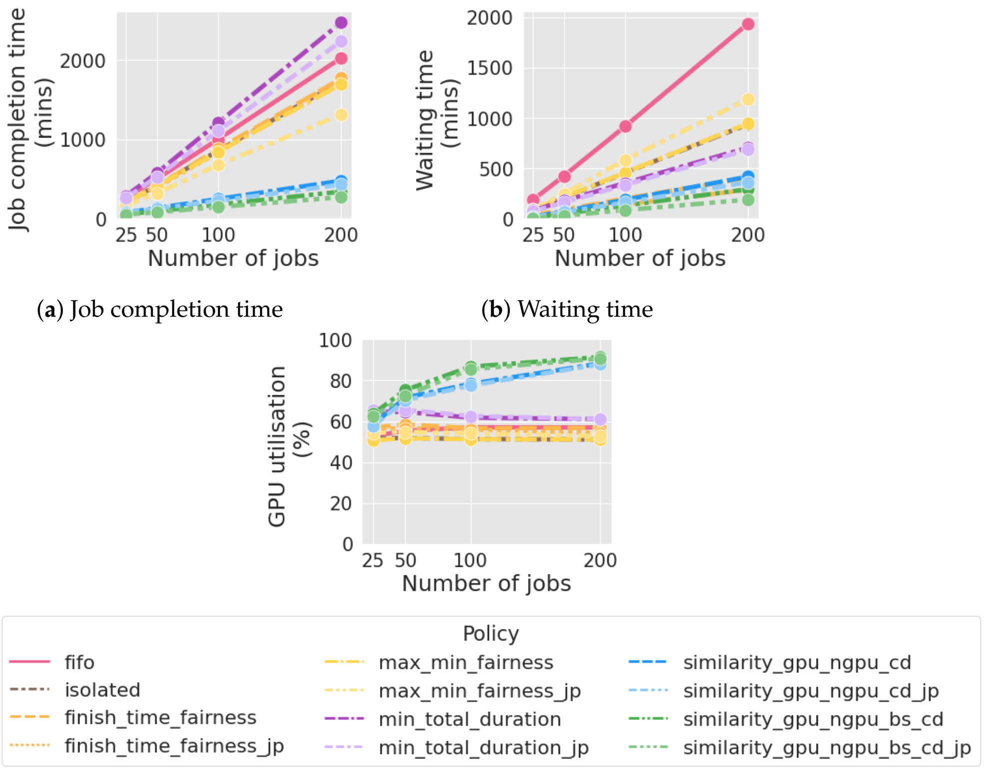


Figure 7. Comparison of makespan of our experiments to the state of the art in three scenarios of job density with rescheduling every three minutes. The blue and green lines represent our experiments.



(**a**) Job completion time (**b**) Waiting time

(**c**) GPU utilisation

Figure 8. Comparison of several evaluation metrics of our experiments to the state of the art in the scenario that all jobs arrive simultaneously, with rescheduling every three minutes. The blue and green lines represent our experiments, that is, *similarity_gpu_ngpu* and *similarity_gpu_ngpu_bs*, respectively. The other lines represent the state of the art.

The observed improvement in our work can be attributed to taking into account several influential factors regarding the training performance, including GPU architecture, the number of GPUs used, the DL model, and the DL hyperparameter (batch size). The state-of-the-art methods show awareness of the diverse training performance arising from the DL model and GPU architecture combined, but overlook the number of GPUs and batch size in making scheduling decisions.

Table 3. Scheduling policies for comparison.

Policy	Description
fifo	First in, first out (in YARN [39])
isolation	Dominant Resource Fairness [40]
min_total_duration	Gavel to minimise makespan [8]
max_min_fairness	Tiresias [41]
finish_time_fairness	Themis [42]
similarity	Our scheduling policy
similarity_gpu	Decision on GPU architecture
similarity_gpu_ngpu	Decision on GPU architecture and the number of GPUs
similarity_gpu_ngpu_bs	Decision on GPU architecture, the number of GPUs, and the adjustment of batch size

7. Conclusions and Future Works

In conclusion, this work presents a scheduling policy for DL training tasks in a heterogeneous GPU cluster, addressing the variability of training performance. It develops upon the model-similarity-based scheduling policy by implementing a round-based mechanism and job packing. The round-based mechanism empowers the scheduler to periodically adjust scheduling decisions for a given job to optimise training performance. Furthermore, job packing enables the concurrent training of multiple models to improve GPU utilisation, particularly when a GPU is allocated for training a small model. As a result, it mitigates queuing delay during high cluster load.

This work conducts various experiments considering three influential factors on training performance: GPU architecture, the number of GPUs used in training, and batch size. While batch size is typically a user choice, and its adjustment is optional for a scheduler to improve cluster efficiency, this work demonstrates improvements when adjusting the batch size along with a selection of GPU architecture and the number of GPUs used in training. When comparing our work to the state of the art, our work outperforms across all evaluation metrics, particularly under high cluster loads. We conclude that the model-similarity-based scheduling policy is more effective when implemented with the round-based mechanism and job packing than when implementing the model-similarity-based scheduling policy alone.

Future works can be extended by improving the round-based mechanism to address its current drawbacks, where resources may idle if a job is completed before the round ends. One potential solution to address this drawback is to implement a predictive approach at the beginning of each round. This approach involves assessing whether the jobs scheduled for the current round are likely to finish before the round ends. If such a scenario is anticipated, the scheduler can pre-determine the scheduling of a job in the queue that is compatible with the allocated resources of the potentially finished job.

Author Contributions: Conceptualization, P.T. and J.S.; Methodology, P.T. and J.S.; Validation, P.T.; Investigation, P.T.; Data curation, P.T.; Writing—original draft, P.T.; Writing—review & editing, K.L., F.L., A.G., J.S. and P.B.; Visualization, P.T.; Supervision, K.L., F.L., A.G., J.S. and P.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analysed in this study. This data can be found here: https://gitlab.uni.lu/pthanapol/model-similarity-based-scheduling-policy.

Acknowledgments: The authors are grateful for the HPC of the University of Luxembourg (ULHPC) and Grid'5000, which provided computing resources throughout this research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: http://www.deeplearningbook.org (accessed on 9 March 2024).
2. Ni, J.; Young, T.; Pandelea, V.; Xue, F.; Cambria, E. Recent advances in deep learning based dialogue systems: A systematic survey. *Artif. Intell. Rev.* **2023**, *56*, 3055–3155. [CrossRef]
3. Adate, A.; Tripathy, B.K. A Survey on Deep Learning Methodologies of Recent Applications. In *Deep Learning in Data Analytics: Recent Techniques, Practices and Applications*; Springer International Publishing: Cham, Switzerland, 2022; pp. 145–170. [CrossRef]
4. Hu, Q.; Sun, P.; Yan, S.; Wen, Y.; Zhang, T. Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Association for Computing Machinery, SC'21, Denver, CO, USA, 17–22 November 2021.
5. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A System for Large-Scale Machine Learning. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16, Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
6. Yu, G.X.; Gao, Y.; Golikov, P.; Pekhimenko, G. Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training. In Proceedings of the USENIX Annual Technical Conference (USENIX ATC 21). USENIX Association, Vitual Online, 14–16 July 2021; pp. 503–521.
7. Thanapol, P.; Lavangnananda, K.; Leprévost, F.; Schleich, J.; Bouvry, P. Scheduling Deep Learning Training in GPU Cluster Using the Model-Similarity-Based Policy. In Proceedings of the Intelligent Information and Database Systems, Singapore, Phuket, Thailand, 24–26 July 2023; pp. 363–374.
8. Narayanan, D.; Santhanam, K.; Kazhamiaka, F.; Phanishayee, A.; Zaharia, M. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, Berkeley, CA, USA, 4–6 November 2020; pp. 481–498.
9. Xiao, W.; Bhardwaj, R.; Ramjee, R.; Sivathanu, M.; Kwatra, N.; Zhang, Q.; Yang, F.; Zhou, L. Gandiva: Introspective cluster scheduling for deep learning. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, USA, 8–10 October 2018; pp. 595–610.
10. Peng, Y.; Bao, Y.; Chen, Y.; Wu, C.; Guo, C. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In Proceedings of the 13th EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–14. [CrossRef]
11. Justus, D.; Brennan, J.; Bonner, S.; McGough, A.S. Predicting the computational cost of deep learning models. In Proceedings of the IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 3873–3882.
12. Yang, G.; Shin, C.; Lee, J.; Yoo, Y.; Yoo, C. Prediction of the resource consumption of distributed deep learning systems. *Proc. Acm Meas. Anal. Comput. Syst.* **2022**, *6*, 1–25.
13. Shin, C.; Yang, G.; Yoo, Y.; Lee, J.; Yoo, C. Xonar: Profiling-based Job Orderer for Distributed Deep Learning. In Proceedings of the IEEE 15th International Conference on Cloud Computing (CLOUD), Barcelona, Spain, 10–16 July 2022; pp. 112–114. [CrossRef]
14. Gong, Y.; Li, B.; Liang, B.; Zhan, Z. Chic: Experience-driven scheduling in machine learning clusters. In Proceedings of the International Symposium on Quality of Service, Phoenix, AZ, USA, 24–25 June 2019; pp. 1–10. [CrossRef]
15. Bao, Y.; Peng, Y.; Wu, C. Deep Learning-based Job Placement in Distributed Machine Learning Clusters. In Proceedings of the IEEE International Conference on Computer Communications (IEEE INFOCOM), Paris, France, 29 April–2 May 2019; pp. 505–513. [CrossRef]
16. Luan, Y.; Chen, X.; Zhao, H.; Yang, Z.; Dai, Y. SCHED^2: Scheduling Deep Learning Training via Deep Reinforcement Learning. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–7.
17. Friesel, D.; Spinczyk, O. Black-box models for non-functional properties of AI software systems. In Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI, Pittsburgh, PA, USA, 16–17 May 2022; pp. 170–180.
18. Peng, Y.; Bao, Y.; Chen, Y.; Wu, C.; Meng, C.; Lin, W. DL2: A deep learning-driven scheduler for deep learning clusters. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 1947–1960. [CrossRef]
19. Yeung, G.; Borowiec, D.; Yang, R.; Friday, A.; Harper, R.; Garraghan, P. Horus: An Interference-Aware Resource Manager for Deep Learning Systems. In *Algorithms and Architectures for Parallel Processing, Proceedings of the 20th International Conference, ICA3PP 2020, New York City, NY, USA, 2–4 October 2020*; Qiu, M., Ed.; Springer International: Berlin/Heidelberg, Germany, 2020; pp. 492–508.
20. Yeung, G.; Borowiec, D.; Yang, R.; Friday, A.; Harper, R.; Garraghan, P. Horus: Interference-Aware and Prediction-Based Scheduling in Deep Learning Systems. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 88–100. [CrossRef]
21. Krizhevsky, A.; Hinton, G. Learning Multiple Layers of Features from Tiny Images. Technical Report. 2009. Available online: https://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf (accessed on 9 March 2024).
22. Narayanan, D.; Harlap, A.; Phanishayee, A.; Seshadri, V.; Devanur, N.R.; Ganger, G.R.; Gibbons, P.B.; Zaharia, M. PipeDream: Generalized pipeline parallelism for DNN training. In Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP'19, New York, NY, USA, 27–30 October 2019; pp. 1–15. [CrossRef]
23. Krislock, N.; Wolkowicz, H. *Euclidean Distance Matrices and Applications*; Springer: Berlin/Heidelberg, Germany, 2012.
24. Thompson, K.P. The nature of length, area, and volume in taxicab geometry. *Int. Electron. J. Geom.* **2011**, *4*, 193–207.

25. Han, J.; Kamber, M.; Pei, J. *Chapter 2—Getting to Know Your Data*, 3rd ed.; The Morgan Kaufmann Series in Data Management Systems; Morgan Kaufmann: Boston, MA, USA, 2012; pp. 39–82. [CrossRef]
26. Yabuuchi, H.; Taniwaki, D.; Omura, S. Low-latency Job Scheduling with Preemption for the Development of Deep Learning. In Proceedings of the USENIX Conference on Operational Machine Learning 2019 (OpML 19), Santa Clara, CA, USA, 20 May 2019; pp. 27–30.
27. Go, Y.; Shin, C.; Lee, J.; Yoo, Y.; Yang, G.; Yoo, C. Selective Preemption of Distributed Deep Learning Training. In Proceedings of the IEEE 16th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 2–8 July 2023; pp. 175–177.
28. Sherstinsky, A. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Phys. D Nonlinear Phenom.* **2020**, *404*, 132306. [CrossRef]
29. Wu, J. Introduction to convolutional neural networks. *Natl. Key Lab Nov. Softw. Technol. Nanjing Univ. China* **2017**, *5*, 495.
30. Chollet, F. Keras—An Open-Source Neural-Network Library Written in Python. 2015. Available online: https://github.com/fchollet/keras (accessed on 9 March 2024).
31. Margery, D.; Morel, E.; Nussbaum, L.; Richard, O.; Rohr, C. Resources Description, Selection, Reservation and Verification on a Large-scale Testbed. In Proceedings of the 9th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TRIDENTCOM), Guangzhou, China, 5–7 May 2014.
32. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
33. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016.
34. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
35. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVF), Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520. [CrossRef]
36. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
37. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
38. Błażewicz, J.; Ecker, K.H.; Pesch, E.; Schmidt, G.; Sterna, M.; Weglarz, J. *Handbook on Scheduling: From Theory to Practice*; Springer: Berlin/Heidelberg, Germany, 2019.
39. Douglas, C.; Lowe, J.; Malley, O.O.; Reed, B. Apache Hadoop YARN: Yet Another Resource Negotiator. In Proceedings of the 4th Annual Symposium on Cloud Computing, Santa Clara, CA, USA, 1–3 October 2013; pp. 1–16.
40. Ghodsi, A.; Zaharia, M.; Hindman, B.; Konwinski, A.; Shenker, S.; Stoica, I. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11), Boston, MA, USA, 25–27 April 2011.
41. Gu, J.; Chowdhury, M.; Shin, K.G.; Zhu, Y.; Jeon, M.; Qian, J.; Liu, H.; Guo, C. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), Boston, MA, USA, 26–28 February 2019; pp. 485–500.
42. Mahajan, K.; Balasubramanian, A.; Singhvi, A.; Venkataraman, S.; Akella, A.; Phanishayee, A.; Chawla, S. Themis: Fair and Efficient GPU Cluster Scheduling. In Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), Santa Clara, CA, USA, 25–27 February 2020; pp. 289–304.