ISSN 1846-6168 (Print), ISSN 1848-5588 (Online)
https://doi.org/10.31803/tg-20240112104444

Preliminary communication
Received: 2024-01-12, Accepted: 2024-05-21

Efficient Deep Learning Job Allocation in Cloud Systems by Predicting Resource Consumptions including GPU and CPU

Abuda Chad Ferrino, Tae Young Choe*

Abstract: One objective of GPU scheduling in cloud systems is to minimize the completion times of given deep learning models. This is important for deep learning in cloud environments because deep learning workloads require a lot of time to finish, and misallocation of these workloads can create a huge increase in job completion time. Difficulties of GPU scheduling come from a diverse type of parameters including model architectures and GPU types. Some of these model architectures are CPU-intensive rather than GPU-intensive which creates a different hardware requirement when training different models. The previous GPU scheduling research had used a small set of parameters, which did not include CPU parameters, which made it difficult to reduce the job completion time (JCT). This paper introduces an improved GPU scheduling approach that reduces job completion time by predicting execution time and various resource consumption parameters including GPU Utilization%, GPU Memory Utilization%, GPU Memory, and CPU Utilization%. The experimental results show that the proposed model improves JCT by up to 40.9% on GPU Allocation based on Computing Efficiency compared to Driple.

Keywords: cloud computing; convolutional neural network; deep learning; GPU job scheduling; performance estimation

1 INTRODUCTION

Deep learning is used in various fields such as Chat-GPT [1] that generates responses to user inquiries. It is also used for structure design to predict equipment failure [2], automatic vehicle incident model [3, 26], and video anomaly detection [4]. With the success of deep learning used in various fields of study, the complexity of these deep learning models also increases as a higher number of parameters and layers yield better accuracy.

As proposed in the Convolutional Neural Network (CNN) used in [4], the stacked 3×3 convolution kernels from VGG16 have been increased up to 7×7 convolution kernels. The increase in parameters and layers of a neural network model also increases the training time needed to train a given model to obtain a desirable accuracy. The trend in the increased complexity of deep learning models not only increases training times but also increases the hardware requirements of training such models. Scheduling in deep learning is to allocate computational resources and jobs that will optimize and accelerate model training. One of the difficulties of scheduling jobs for deep learning models is that there is a diverse amount of model architectures, resulting in different hardware requirements to train such models. Some of these models require high GPU performance and some other models require higher CPU performance.

As a result, scheduling mechanisms must be able to adapt to these different model architectures and allocate the jobs accordingly based on the specific needs of a given model. Previous work [5] claimed that operations used in a model have different impacts on execution time, but in this work, they only calculated the execution time of each operation and scaled it into an entire iteration. Meanwhile [6] focused on the 'features' derived from the network being trained, the data being trained, and the hardware that the model is being trained on, for evaluating resource consumption to estimate execution time. However, this work only utilized a VGG16 model to test their model. On the other hand, others focused only on the Central Processing Unit (CPU) [7] to estimate the execution time of Convolutional Neural Networks (CNN) on a single CNN model with three different architectural sizes. Lastly, [8] introduced Graph Neural Network (GNN) and utilized Graphics Processing Unit (GPU) and network parameters for resource consumption prediction to estimate the resource consumption based on three prediction targets, for each resource consumption parameter totaling up to 12 total prediction targets, while it considers the entire graph as an input, which involves all the operations found within the graph, they only focused on GPU parameters such as GPU Utilization% and GPU Memory Utilization%.

In this paper, resource consumption and execution time prediction is explored by evaluating the different Resource Consumption parameters that are obtained from the GPU, and CPU, during training, to create a model that will be utilized to predict resource consumption and execution time for predicting the GPU allocation of a given Deep Learning task, and lastly, applying a scheduling algorithm, such as First-In First-Out (FIFO) and Shortest Job First (SJF) to test the effectiveness of this approach in real-world applications. This paper also introduces a new approach for predicting GPU Job allocation by using computing efficiency, which is derived from the resource consumption prediction parameters. To achieve these objectives this study aimed to:

- Create a dataset with four resource consumption parameters (GPU Utilization, GPU Memory Utilization, CPU Utilization, and GPU Memory) and four prediction targets for each parameter (Average Burst Time, Average Idle Time, Average Peak Consumption, Execution Time) which amounts to 16 prediction metrics.
- Develop a model that will not only predict resource consumption but also the execution time of a given input which can be used for scheduling jobs.
- Evaluate the performance of prediction targets as parameters for GPU allocation and its effects on job scheduling using FIFO and SJF scheduling policies.

2 BACKGROUND AND RELATED WORKS

2.1 Job Profiling

Previous works such as [6] dissected a given model architecture and utilized what they called Layer Features, which includes batch size, optimizer, and activation function. In addition, they also included layer-specific features, such as Convolutional features, pooling features, and so on, which are mostly found on a CNN model, in this example they only used VGG16. They also included some GPU hardware specification as part of their training features to create a profiler that will predict the execution time of each layer, to predict the full model execution time. However, this is very limited to models that are like VGG16. This approach requires dissecting a target model and figure out all the Layer Features, Convolutional features, and GPU hardware specifications, which will be difficult to implement on different neural network models due to the varying model architectures and operations used in each model.

Another work involves the use of a CPU [7] instead of a GPU for profiling CNNs of small, medium, and large CNN architectures, with varying amount of convolutional layers. They mainly utilized forward propagation and backward propagation per image for their measurements to predict the execution time, which is also found in [9]. Contrary to [7], in [9], they used a multi-GPU approach and were limited by small-scale CNN model architectures. While these works predicted execution time, it only covered very specific neural network models

Lastly, in [8] they profiled an input task, as shown in Fig. 1, and utilized TensorFlow to convert it to a graph, which produces an adjacency matrix and feature matrix. This job profiler produces three outputs that characterize resource consumption: active time, idle time, and average peak consumption. With these, they were able to estimate the resource consumption of various models with different hyperparameter settings.

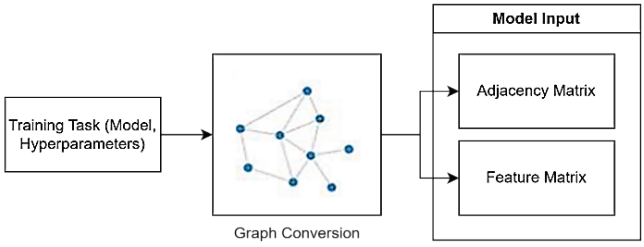

Figure 1 Input task conversion to graph

2.2 Resource Consumption Parameters

The system used in this paper uses a job profiler based on Driple [10] to create a model that will predict resource consumption by the usage of resource consumption parameters on GPU, and CPU along with execution time, which will then be used for scheduling. While other works only used GPU, CPU, or multi-GPU hardware settings. This paper proposes to utilize both GPU and CPU parameters, namely GPU Utilization%, GPU Memory, GPU Memory Utilization%, and CPU Utilization%. The GPU parameters are obtained via the usage of the NVIDIA System Management Interface (nvidia-smi) library [11] and CPU parameters were obtained using psutil library which calculates the CPU Utilization% for all cores. These parameters are profiled in an interval of 1/6 per second. Previous work [8], utilized only GPU Utilization% and GPU Memory Utilization for GPU parameters.

However, in this approach the prediction output of these input parameters will be used for GPU allocation of jobs, hence the addition of GPU Memory and CPU Utilization% on the resource consumption parameters are proposed. As these parameters can also affect the performance of training a model as larger architectures would prefer bigger amounts of GPU Memory for optimal performance, in some cases, it will not be able to train the model at all [12]. Moreover, based on initial experiments using the same hyperparameter settings of a VGG model on 3 machines, as seen in Table 1, the RTX2070 machine simulated as a low-performance system shows that the resource usage between the other two machines is significant. This was identified as a CPU bottleneck that can occur on lower-performance systems, which makes the training time longer and makes it difficult to perform job profiling during training as shown on the vgg19 model.

Table 1 Profiling of average GPU utilization (%), GPU memory (MB) for VGG models

Dataset	GTX1080	RTX2070	Titan X	Model
ImageNet	98.84%, 7840.87	26.59%, 5738.25	98.27%, 11926.75	vgg11
	99.18%, 7856.85	16.16%, 5622.33	98.74%, 11929.67	vgg16
	89.95%, 7712.80	-	98.88%, 11930.06	vgg19

2.3 Evaluation of Resource Consumption

In this work, similar output parameters were utilized. However, in the context of a scheduler, the amount of resources consumed by a job is of importance. Hence, the output parameters are proposed to be defined as follows: burst time is the upper half of the median for each resource type, while idle time is defined by the lower half of the median for each resource type.

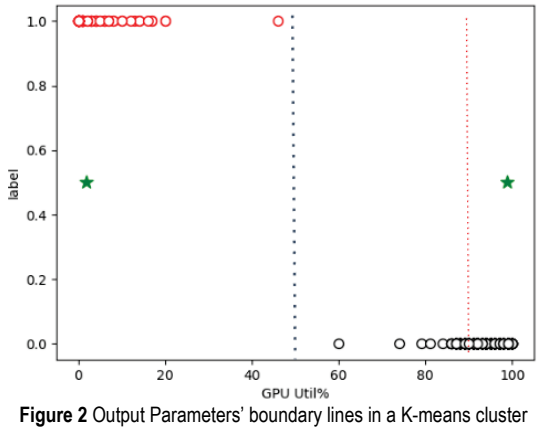

Figure 2 Output Parameters' boundary lines in a K-means cluster

For example, for a workload having a GPU utilization between 0-60%, the burst time is 31-60%, while idle time is 0-30% utilization. For the peak consumption, it considers the

maximum capacity of each resource type, which is > 90% of its maximum value. In terms of GPU utilization, it will be data that's > 90% utilization and for memory, it will be > 90% of the total available memory in the GPU. These are represented by the black dotted line representing the median and red dotted lines representing the boundary for peak consumption data points on the right side, as seen on an example benchmark in Fig. 2.

By measuring GPU Memory Utilization%, GPU Utilization%, GPU Memory, and CPU Utilization% for each workload, they are converted into these three parameters using these criteria. These are then grouped together by their respective hardware setup to create a dataset, which will be used to train a model to produce the predicted resource consumption on a given GPU or hardware setup. These predicted values are then used for GPU allocation of succeeding deep learning workloads.

For scheduling, the execution time of a workload also helps with decision-making for job allocation. Hence, in addition to the mentioned parameters, the execution time will also be included as part of the input and output parameters. The input execution time will be measured during the job profiling phase, and it will be normalized based on the maximum and minimum execution time values found on the dataset. The output execution time will be part of the prediction targets along with the three mentioned parameters, burst time, idle time, and average peak consumption.

Therefore, this paper profiles five different resource consumption parameters, GPU Utilization%, GPU Memory Utilization%, GPU Memory, CPU Utilization%, and Wall-Clock Execution Time. Among the hardware resource consumption parameters, each will have four prediction output targets (burst time, idle time, average peak consumption, and execution time), having a total of 16 prediction output parameters, as shown in Tab. 2.

Table 2 Resource consumption parameters for profiling

Proposed Parameters	Prediction Targets
GPU Utilization%	Burst Time, Idle Time, Average Peak Consumption, Execution Time
GPU Memory Utilization%	Burst Time, Idle Time, Average Peak Consumption, Execution Time
GPU Memory	Burst Time, Idle Time, Average Peak Consumption, Execution Time
CPU Utilization%	Burst Time, Idle Time, Average Peak Consumption, Execution Time
Wall Clock Execution Time	*-Part of the prediction targets above-*

2.4 Job Scheduling

Previous works on job scheduling implemented a generalized wide range of scheduling policies [13], such as First-In-First-Out (FIFO) and Shortest Job First (SJF) policies, to measure the makespan, which is the amount of time it takes to complete all the jobs scheduled, and job completion time for deep learning training workloads. While Gavel focused on job allocation based on policies, and to improve makespan and JCT, they did not use a prediction model and they did not consider hardware resource consumption parameters.

Another scheduler focused on a directed acyclic graph (DAG) [14], which involves adding idle time in between jobs to decrease the average job completion time of the workloads. While they used graphs as inputs, it did not involve deep learning workloads. The proposed model has additional input parameters and prediction targets compared to the previous work [8]. Specifically, the proposed model intends to profile jobs based on GPU, and CPU parameters, such as GPU Utilization%, GPU Memory Utilization%, GPU Memory, and CPU Utilization%, while also including Execution Time.

This is different compared to the previous model which only uses GPU Resources such as, GPU Utilization%, and GPU Memory Utilization%, and network parameters. However, in this research, network parameters were not considered since deep learning workloads are allocated on a single machine after GPU Allocation and network parameters does not impact the performance of training. This research introduces computing efficiency that is derived from the resource consumption prediction to schedule jobs. The performance of computing efficiency is evaluated by measuring the makespan and average job completion time using FIFO and SJF policies. A similar approach is also applied to job allocation based on predicted execution time.

2.5 Challenges of GPU Scheduling

GPU Scheduling is the process of allocating and managing GPU resources to various computational tasks or processes. Ref. [17] addressed various difficulties in GPU scheduling especially in larger scale systems where the workload has various computational requirements, which was divided into two types which were identified as High-GPU Tasks and Low-GPU Tasks.

High-GPU Tasks involves Natural Language Processing (NLP) [18] with advanced language models, Image classification with a large output such as a modified ResNet model [19]. High GPU tasks could potentially consume the entire GPU resource usage while having low CPU resource utilization.

For Low-GPU Tasks it was discovered that these tasks spend a considerable amount of time on CPUs for data processing. Some of these tasks involve click-through rate (CTR) [26] prediction models, Graph Neural Network (GNN) [20] Training models and Reinforcement learning. Among these tasks the GPU is underutilized on Low-GPU tasks mainly because of the CPU bottleneck where the CPU limits the amount of GPU resource that can be used by the model, leaving the GPU usage underutilized [28].

As such, it was mentioned that considering a multi-resource scheduler would be preferable to alleviate the scheduling difficulties. Hence, the proposed model intends to profile jobs based on GPU, and CPU parameters, such as GPU Utilization%, GPU Memory Utilization%, GPU Memory, and CPU Utilization%, while also including Execution Time. This is different compared to the previous model [8] which only uses GPU Resources such as, GPU Utilization%, and GPU Memory Utilization%, and network parameters. However, in this research, network parameters were not considered since deep learning workloads are allocated on a single machine after GPU Allocation and network parameters does not impact the performance of training. This research introduces computing efficiency that

is derived from the resource consumption prediction to schedule jobs. The performance of computing efficiency is evaluated by measuring the makespan and average job completion time using FIFO and SJF policies. A similar method is also applied to job allocation based on predicted execution time. This approach introduces a way of scheduling deep learning workloads by only specifying the model architecture.

3 METHODOLOGY

3.1 Job Profiler

The system workflow consisted of two major parts, namely the job profiler and the job scheduler. The job profiler is responsible for the creation of the proposed prediction model that will be utilized to help the schedule decide which GPU to allocate for a given job. First, the job profiler, as seen in Fig. 3, takes in a training workload, which is represented by a given neural network model, the dataset to be used, and its hyperparameters such as batch size, epochs, and optimizer.

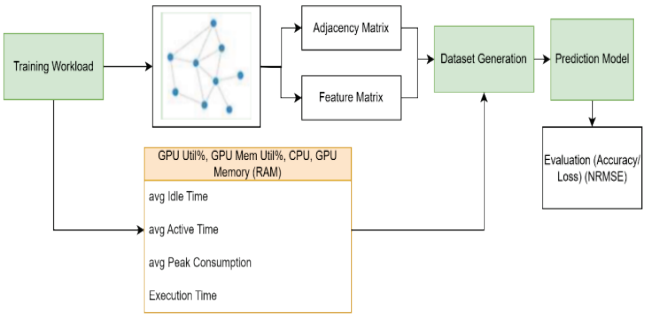

Figure 3 Job profiler workflow

The training workload consists of all the combinations of these settings, as shown in Tab. 3, separating generative neural networks with convolutional neural networks, which totals up to 432 training workloads for a given GPU.

Table 3 Hyperparameter settings

Datasets	Optimizer	Batch Size	Epoch	Models
ImageNet, Cifar10	SGD, Adam	8, 16, 32, 64	10, 20	vgg11, vgg16, vgg19, lenet, googlenet, densenet40-k12, densenet100-k12, inception3,inception4, resnet20, resnet50, resnet101
MNIST	SGD, Adam	8, 16, 32, 64	10, 20	dcgan, conditional_gan, cvae

When starting a training workload, the model being trained is converted into a graph, producing an adjacency matrix that describes the connectivity between nodes and edges, with a value of 1 if there is a connection and 0 if there is none, while the feature matrix consists of tensor size and node type. The node type is the operations found in each model, such as Conv2D, that is converted into a float number by implementing frequency encoding, which estimates the number of occurrences of a given operation found in a dataset. This implementation is similar to the one used in Driple [10].

At the same time, the resource consumption of the system was profiled based on four parameters: GPU Memory Utilization%, GPU Utilization%, GPU Memory, and CPU Utilization% at a rate of 1/6 per second, including the total Execution Time of a given workload. These parameters are segregated into Burst data points and Idle data points using Kmeans, shown previously in Fig. 2, to label them into two clusters according to their resource consumption. These labeled data points are then used to obtain the data for average burst time, average idle time, and average peak consumption rate. Once the data for all parameters are converted, it is then combined with the adjacency matrix, feature matrix, and execution time to create a single entry into the dataset which also includes, the GPU type, the dataset used for training, and the hyperparameters used for the training. Given the settings in Tab. 3, a dataset will contain 384 training workloads for each GPU that will be used for training the proposed model. In this study, three GPUs were used for job profiling which means that there were three datasets that were created.

Once the necessary datasets have been created, with the help of the Driple model [10], these datasets were trained and evaluated based on the loss function Mean Square Error (*MSE*) where N is the number of data, y is the ground truth and $\hat{y}$ is the predicted value.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2. \quad (1)$$

3.2 Job Scheduler

Once the prediction model is trained, the best model will be used by the job scheduler, as shown in Fig. 4. The job scheduler takes in a deep learning training workload as an input, containing the model architecture, and hyperparameter settings. Similarly, since the prediction model takes adjacency matrices and feature matrices as input, the input workload must be converted to a graph to obtain these matrices.

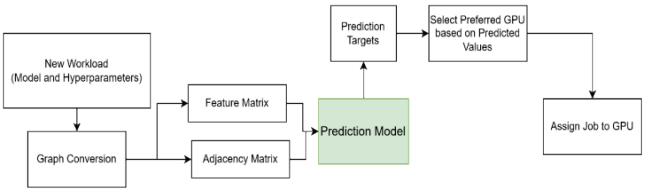

Figure 4 Scheduler workflow

After the conversion, the matrices will then be fed as an input to the prediction model, and the prediction model will produce 16 prediction targets, one for each resource consumption type, namely Average Burst Time, Average Idle Time, Average Peak Consumption, and Execution Time. Among these prediction targets, the scheduler will decide which GPU should a given job be allocated to by measuring the execution time for each dataset and measuring the

computing efficiency ($\hat{y}_{\text{eff}}$) by obtaining a summation of the quotient of the predicted average peak consumption rate ($\hat{y}_{\text{peak}}$) divided by the predicted average burst time ($\hat{y}_{\text{burst}}$) for all resource consumption parameters.

$$\hat{y}_{\text{eff}} = \sum_{i=1}^{N} \frac{\hat{y}_{\text{peak}}}{\hat{y}_{\text{burst}}}. \quad (2)$$

For the execution time, the lower the value, the better the performance of a given workload, hence, GPUs with the lowest predicted execution time will be preferred when scheduling this task. On the other hand, for computing efficiency, the higher the average peak consumption-average burst time ratio translates to the computing resources being fully utilized by the workload while having minimal idle time. Therefore, when considering computing efficiency, the GPU with the highest ratio will be preferred when scheduling a given task. This will be done for all available jobs in the scheduler until all the jobs are assigned.

The performance of the scheduler will be measured by calculating the average job completion time, and the makespan of the schedule. The scheduler workload that will be used for the experiments will be using the combinations of the hyperparameter settings in Tab. 4. Since the models trivial, alexnet, resnet32, and gan are not a part of the dataset in which the model was trained on, these models will be considered as unknown inputs. The scheduler workload will involve up to 128 jobs for GPU allocation.

Table 4 Scheduler workload

Datasets	Optimizer	Batch Size	Epoch	Models
ImageNet, Cifar10	SGD, Adam	8, 16, 32, 64	10, 20	trivial, alexnet, resnet32
MNIST	SGD, Adam	8, 16, 32, 64	10, 20	gan

The algorithm for GPU allocation based on execution time as shown in Algorithm 1, jobs are loaded into the queue, and for each job in the queue, it is converted to a graph. Once the conversion is finished it is then loaded into the prediction model along with the GPU dataset to obtain the execution time prediction for a given dataset.

```
queue = get.Jobs()

for job in queue:
        inputs = convert_to_graph(job)

        for gpu in datasets:
                execution_time_pred += prediction_model(inputs,gpu)
job.GPUAllocation(argmin(execution_time_pred))

if SJF:
        for each GPU:
                queue = sort.by(execution_time_pred,ascending)
```

Algorithm 1 Pseudocode for scheduling based on execution time predictions.

Since there are four resource consumption types, GPU Utilization%, GPU Memory Utilization%, GPU Memory, and CPU Utilization%, the prediction model produces an execution time for each resource consumption type. Therefore, the predicted execution time among the four resource consumption types will be added together. From the predictions obtained from all the datasets, the GPU with the lowest predicted execution time will be the designated allocation for the given job. If the scheduling policy is SJF, then the jobs are sorted after allocation on the GPU based on the predicted execution time.

```
queue = get.Jobs()

for job in queue:
        inputs = convert_to_graph(jobs)

        for gpu in datasets:
                for each resource_consumption_type:
                        resource_consumption_pred = prediction_model(inputs,gpu)
                        comp_eff += resource_consumption_pred[peak] / resource_consumption_pred[burst]
job.GPUAllocation(argmax(comp_eff))

if SJF:
        for each GPU:
                queue = sort.by(comp_eff,ascending)
```

Algorithm 2 Pseudocode for scheduling based on computing efficiency.

On the other hand, GPU allocation based on computing efficiency as shown in Algorithm 2, similarly, jobs are loaded into the queue, then the jobs are converted into graphs, and loaded into the prediction model with the GPU dataset. However, this time, there are four resource consumption types, GPU Utilization%, GPU Memory Utilization%, GPU Memory, and CPU Utilization%, which produces four prediction targets each, the average burst time, average idle time, average peak consumption, and execution time.

For the computation of computing efficiency, using Eq. (2), the ratio of the predicted peak consumption and predicted burst time for each resource type will be combined to a single dataset. After going through all datasets, the highest computing efficiency score will be the designated GPU allocation of a given job. If the scheduling policy is SJF, the jobs will be sorted after allocation based on their computing efficiency scores.

4 EXPERIMENTS AND RESULTS

4.1 Experiment Setup

The experiment was applied on three different servers, which are GTX 1080 8GB, RTX 2070 8GB, and Titan X 12GB, as shown in Tab. 5. The operating system used was Ubuntu 18.04, while using Tensorflow 2.5, and CUDA 11.2 to train a workload on three datasets, which are ImageNet, Cifar10, and MNIST dataset.

Table 5 Machine specifications

GPU Type	CPU Model	GPU Memory Capacity	Memory (RAM)
GTX1080	Intel Core i7-4790K	8 GB	32GB
RTX2070	AMD Ryzen 5 3600	8 GB	32GB
TitanX	Intel Core i7-6900k	12 GB	64GB

Training setting is specified by using a combination of different optimizers, batch sizes, epochs, and models as shown in Tab. 3, with the help of TensorFlow benchmark library [15], Deep Convolutional Generative Adversarial Network (DCGAN) model [23][27], Conditional Generative Adversarial Network (Conditional GAN) [24], Convolutional Variational AutoEncoder (VCAE) [25] were

also included. For training the prediction model, recommended setup by [8] was used and executed on PyTorch 1.8.1

4.2 Hyperparameter Analysis

To better understand the effects of each hyperparameter to the execution time, along with the hardware specifications, Batch Size, Epochs, Optimizers, and GPU Execution times were analyzed as shown in Figs. 5, 6, and 7. It can be inferred from the epoch comparison that for the settings used in this experiment, when the number of epochs is doubled, the execution time is also doubled. On the other hand, as the batch size increases, the execution time decreases, and the improvement is approximately 20% from batch size 8 to batch size 16. However, as the batch size continues to increase, the difference in execution time becomes minimal as seen on batch size 32 and batch size 64.

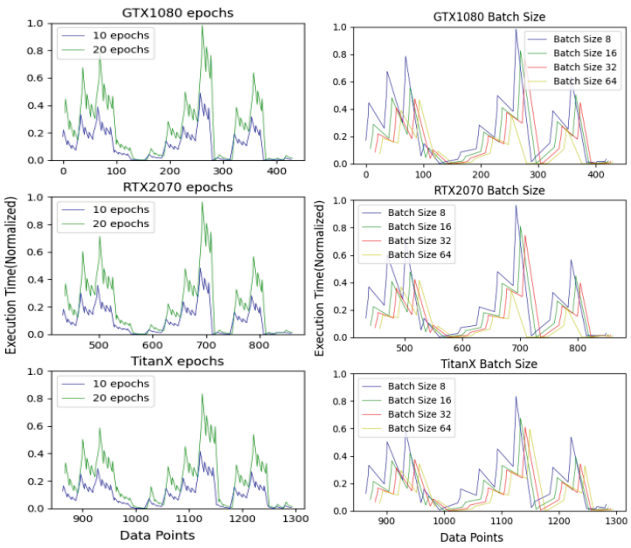

Figure 5 Execution times based on epochs and batch sizes among the 3 GPU machines

For the optimizers, the difference in execution time is very minimal as shown in Fig. 6. Hence, when considering the execution time of a model based on hyperparameters, epochs and batch size has a higher priority compared to optimizers. Lastly, in Fig. 7, the execution time of these models on three machines for these hyperparameter settings shows that GTX1080 has the highest average execution time, and TitanX has the lowest average execution time. For the graphs shown in Figs. 5, 6, and 7, there are some points where the execution time is 0, this is due to the GPU Memory being insufficient, hence the training model cannot be loaded onto the machine and was unable to train the model. It can be observed in Fig. 7, that for GTX1080, and RTX2070 it was unable to train the models that were configured to be Adam optimizer and batch size 64, while TitanX, having a bigger memory capacity, was able to load and train the model.

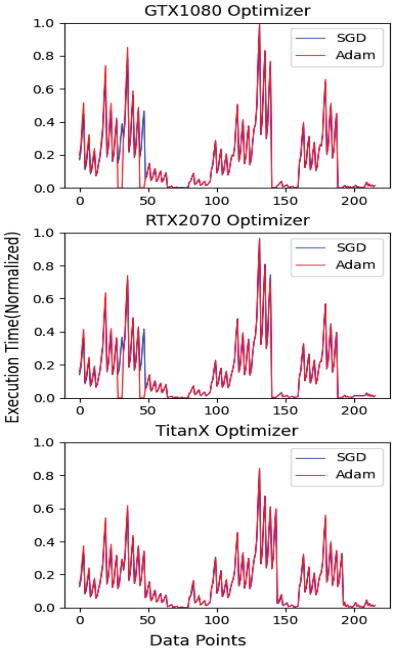

Figure 6 Execution Times based on optimizers among the 3 GPU machines

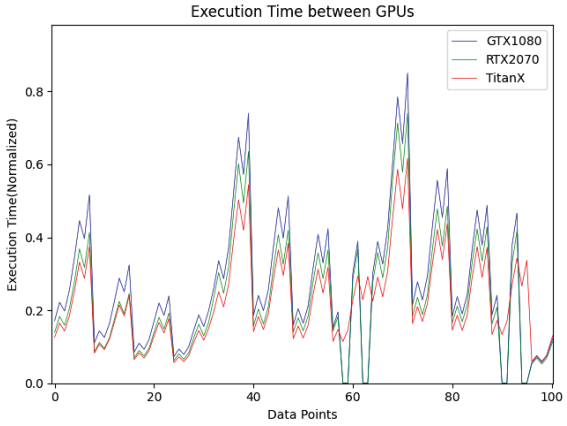

Figure 7 Execution time comparison between 3 GPU machines

4.3 Evaluating the Proposed Model

To test the accuracy of the prediction model, the proposed model was trained on the three GPU datasets generated using the data obtained from the previous section, to evaluate the validation loss using MSE. Driple [8] uses GPU Utilization%, and GPU Memory Utilization%, Network Transmission (Tx), Network Received (Rx) as inputs.

The proposed model changes the input parameters into GPU Utilization%, GPU Memory Utilization%, CPU Utilization%, and GPU Memory, with also the inclusion of Execution Time. Due to the nature of a training workload not being distributed on multiple machines unlike Driple, this approach only considers training the entire model on a single machine. Hence, Network parameters are not considered.

In Tab. 6, from the validation loss perspective, Driple has the best accuracy on the RTX2070 dataset, while the Proposed Model has the best accuracy on the other two datasets, the GTX1080 dataset and TitanX dataset. While the

proposed model also has a higher loss compared to the original model, this is expected due to the increase in parameter inputs, with an increase in total loss of about 6.84%.

Table 6 Validation loss comparison with driple

Model	Validation Loss			Evaluation	
	GTX1080	RTX2070	TitanX	Total Loss	Total Loss vs Original Model
Driple	0.022	0.0072	0.028	0.057	0
Proposed Model	0.017	0.024	0.0199	0.061	+6.84

Considering Validation Loss, it shows that the increase in input parameters increases the complexity of the model, which leads into higher validation loss. Hence, in line with the previous work [8], Transfer Learning is also explored in this research to determine the effects of transfer learning on the validation loss and how it is constructed.

4.4 Transfer Learning

Transfer learning involves using a pre-trained model and using its features as a reference when starting to train a new model. Using these features as a starting point, the training time and epochs required to train a new model will be decreased while also maintaining a similar amount of validation loss. It is also advantageous for models trained on smaller datasets [16], since the datasets used in the experiments only consist of 320 samples for each GPU.

To select which model to use as a pre-trained model for transfer learning, based on the validation loss in Table 6 for the proposed model, the GTX1080 dataset has the lowest validation loss, based on this, it has the best accuracy among the other two datasets. The original model, despite having the best accuracy between all test cases, was not considered because it uses a different set of inputs compared to the proposed model and it would be lacking the features that are necessary to be transferred to the new model that will be trained. Therefore, GTX1080 for the proposed model is selected to be the pre-trained model used for transfer learning.

In Tab. 7, the validation loss is evaluated between the proposed model with Transfer Learning and the model without Transfer Learning. For the validation loss, there is a 5% improvement in the accuracy when Transfer Learning is applied, as the total loss goes down to 0.0578. This is almost comparable to the previous model, Driple, which has a validation loss of 0.057.

Table 7 Validation loss

Model	Validation Loss			Evaluation	
	GTX1080	RTX2070	TitanX	Total Loss	Total Loss vs Proposed Model w/o TL
Proposed Model w/o TL	0.0171	0.0239	0.0199	0.0609	0
Proposed Model w/ TL	0.0174	0.0172	0.0232	0.0578	–5.090311987

Due to the improvement in the validation loss, the proposed model with transfer learning was used as the prediction model that was used for scheduling jobs using FIFO and SJF.

4.5 GPU Allocation for Job Scheduling

For the GPU Allocation of the proposed model, it will be evaluated based on two parameters, Computing Efficiency and Execution Time. Both parameters are outputs of the prediction model and are computed as the sum of these prediction values across all resource consumption parameters. In the case of Computing Efficiency, it is the sum of the ratio of the Average Peak Time and Average Burst Time per resource consumption parameter, while Execution Time is the sum of the predicted Execution Time per resource parameter.

4.5.1 GPU Allocation based on Computing Efficiency

To demonstrate how the GPU Allocating based on Computing Efficiency is determined, 6 jobs found on the dataset as specified in Tab. 8 are used as input to the prediction model. Each job arrives one minute apart from each other.

Table 8 Jobs for FIFO simulation (seconds)

	Optimizer	Batch Size	Epoch	Arrival (minute)
resnet20_cifar10	Adam	8	20	0
densenet40-k12_cifar10	SGD	64	10	1
inception3_imagenet	Adam	16	20	2
vgg11_imagenet	SGD	32	10	3
DCGAN_mnist	Adam	32	20	4
googlenet_imagenet	SGD	64	10	5

To compute the computing efficiency, there will be four different resource consumption parameters per GPU Dataset. Each of these resource consumption parameters has outputs for Average Burst Time, Average Idle Time, and Average Peak Consumption Rate. By utilizing Eq. (2) to obtain the computing efficiency of these output parameters and getting the summation of the ratio of values for four different resource consumption parameters, values shown in Tab. 9 are obtained for each GPU Dataset.

Table 9 Scheduling jobs based on computing efficiency

	GTX1080	RTX2070	TitanX	Allocation	JCT (s)
resnet20_cifar10	1.12	3.12	3.55	TitanX	849.21
densenet40-k12_cifar10	1.91	4.73	3.39	RTX2070	411.09
inception3_imagenet	2.01	4.99	4.51	RTX2070	11,179.09
vgg11_imagenet	2.42	4.76	4.79	TitanX	3,397.21
DCGAN_mnist	1.88	–485.58	–1861.76	GTX1080	248.48
googlenet_imagenet	1.91	–471.49	5.23	TitanX	4358.18

As observed in Tab. 9, the prediction model selects the highest computing efficiency amongst the dataset for GPU Allocation. In the case of DCGAN_mnist, it has the highest

computing efficiency compared to RTX2070 and TitanX, therefore this job is allocated to GTX1080. By following this criteria, the other workloads are allocated to their respective GPUs, and the JCT for each job is measured.

4.5.2 Performance Comparison based on Computing Efficiency

Since Driple [8] uses different input parameters compared to the proposed model, which includes CPU Util%, GPU Memory, and Execution Time, the proposed model and the Driple model were compared based on scheduling based on FIFO and SJF (sorting after job allocation), with the average JCT and makespan as the evaluation parameters. Furthermore, since the previous work does not have the Execution Time as its parameter, the comparison was based on the computing efficiency by using Eq. (2) to calculate the computing efficiency for each prediction target and obtaining the maximum total value for each resource consumption parameter as basis for GPU allocation.

In the case of Driple, the GPU allocation mostly ignored the GTX1080 GPU for this setup, having 0 jobs allocated on this machine, and the rest of the jobs allocated on the other two machines for both FIFO and SJF, as observed in Tabs. 10 and 12. It can be observed that while the makespan is higher on FIFO, the JCT is about 35.44% lower than Driple. However, looking at the makespan for each scheduling algorithm, it can be considered that Driple has the marginal advantage on makespan, even though the proposed model is marginally higher by approximately 0.035% on SJF. In terms of Average JCT, improvements on both models are observed when SJF is applied, with the proposed model 40.9% lower on average JCT on the SJF scheduling algorithm compared to Driple.

Table 10 FIFO based on computing efficiency (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
Driple	0	13814.65	20354.29	18719.38	**35120.41**
Proposed Model	3627.14	11135.98	19748.27	**12084.68**	35929.98

Table 11 FIFO Job Distribution based on computing efficiency

	GTX1080	RTX2070	TitanX
Driple	0 (0%)	10 (7.81%)	118 (92.19%)
Proposed Model	32 (25%)	54 (42.19%)	42 (32.81%)

Table 12 SJF based on computing efficiency (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
Driple	0	13134.74	13791.36	13627.21	34773.36
Proposed Model	3137.301	5941.055	14526.52	8057.22	34785.50

It is also observed that the job distribution does not change from FIFO to SJF since SJF is only reordering the jobs after allocating it to a designated GPU as shown in Tab. 13. Furthermore, since the input consists of training workloads, it can be seen in Tab. 10 for Driple that despite having only 10 jobs allocated to RTX2070, it has way higher JCT Compared to the Proposed Model having 54 jobs allocated to the same GPU. This shows that the training time of training workloads varies a lot depending on the parameters and architecture and reordering them and allocating them correctly improves the JCT of the given jobs.

Table 13 SJF Job Distribution based on computing efficiency

	GTX1080	RTX2070	TitanX
Driple	0 (0%)	10 (7.81%)	118 (92.19%)
Proposed Model	32 (25%)	54 (42.19%)	42 (32.81%)

4.5.3 GPU Allocation based on Execution Time

As Driple does not contain Execution Time, the GPU allocation based on Execution Time are only observed using the proposed model based on FIFO and SJF scheduling algorithms. Using the same set of jobs found in Table 8, the prediction model will produce a predicted execution time value for each resource consumption parameter, up to a total of four predicted values, for each GPU dataset. The predicted values will be summed up together per GPU Dataset as shown in Tab. 14. However, for allocating jobs based on execution time, the lowest predicted execution time for each dataset is considered. Hence, the GPU allocation are decided based on this criterion.

Table 14 FIFO based on predicted execution time (seconds)

	GTX1080	RTX2070	TitanX	Allocation	JCT (s)
resnet20_cifar10	–0.017	–0.036	–0.045	TitanX	819.21
densenet40-k12_cifar10	0.158	0.256	0.265	GTX1080	485.49
inception3_imagenet	0.82	1.52	2.51	GTX1080	13979.41
vgg11_imagenet	0.89	1.64	2.46	GTX1080	17452.4
DCGAN_mnist	1105.15	8929.84	17820.21	GTX1080	17700.88
googlenet_imagenet	1.21	2.22	3.11	GTX1080	19228.15

Contrary to the FIFO Allocation based on computing efficiency, the allocation based on Execution Time ended up assigning most of the selected jobs on GTX1080, which led to a very high JCT for most jobs, while also the RTX2070 not being selected for any of the given jobs for allocation. Based on the results from the allocation based on computing efficiency, shown in Tab. 9 and allocation based on execution time, shown in Tab. 14, it can be inferred that the Allocation performs way better when utilizing computing efficiency for the given jobs.

4.5.4 Performance Evaluation based on Execution Time

This time, the proposed model based on execution time prediction was observed for FIFO and SJF algorithm. As shown in Tab. 15, the average JCT of the proposed model increases by approximately 8.01% when using SJF scheduling compared to FIFO. However, the makespan decreases by approximately 1.09% when using SJF scheduling. Comparing this with the prediction results based on computing efficiency, the average JCT and makespan is increased by up to 124.87% an 23.13%, respectively.

Table 15 Scheduling based on execution time prediction (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
Proposed Model (FIFO)	11054.36	2361.27	23781.35	16665.91	45747.11
Proposed Model (SJF)	14445.05	1911.83	24226.93	18118.01	45252.55

Table 16 Job Distribution based on execution time

	GTX1080	RTX2070	TitanX
Proposed Model (FIFO)	48 (37.5%)	14 (10.94%)	66 (51.56%)
Proposed Model (SJF)	48 (37.5%)	14 (10.94%)	66 (51.56%)

As seen in the GPU allocation based on Execution Time and Computing Efficiency, the scheduler generally performed better on average JCT and Makespan, when based on computing efficiency, while using the proposed model. In contrast to Tabs. 11 and 13, it can be seen in Tab. 13 that jobs with lower JCT are allocated to RTX2070 and majority of the jobs with higher JCT were allocated to GTX1080 and TitanX.

4.6 Ablation Study

To understand the effects of the resource consumption parameters on the GPU allocation based on Average JCT and Makespan, two scenarios were considered by utilizing the same jobs as shown in Tab. 4. First, the resource consumption parameters are separated from each other and treated as an individual parameter for GPU Allocation. While the other scenario involves taking away one of each parameter and observing its effects on Average JCT and Makespan.

4.6.1 Resource Consumption Prediction of Individual Parameters

As shown in Tab. 17, utilizing only one of each resource consumption parameters for scheduling jobs with FIFO and SJF were observed. It shows that GPU Memory Utilization% does not consider GTX1080 for allocation while GPU Memory does not consider TitanX.

For each resource consumption parameter, the Average JCT and Makespan has seen improvements when SJF is applied, except for GPU Memory Utilization%, where it caused the Average JCT to increase by 5.20%. On the other hand, the CPU Utilization% has the lowest average JCT which is comparable to the output of the entire proposed model on Table 11. Furthermore, the makespan for both CPU Utilization% and GPU Memory is also lower than the entire proposed model.

The job distribution in Tab. 18 shows that the number of jobs allocated to a given GPU does not directly represent the JCT of a given job. This is due to the varying nature of the JCT of a given training workload which depends on the hyperparameter settings and model architecture. Therefore, despite having more jobs allocated to RTX2070 on GPU Memory Utilization%, and GTX1080 for CPU Utilization%, the other GPUs that have the workload with higher execution time will have the higher JCT for both cases.

Table 17 Scheduling based on computing efficiency using individual parameters (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
GPU Memory Utilization% (FIFO)	0	10625.55	24724.39	16973.80	46014.70
GPU Memory Utilization% (SJF)	0	12004.40	25492.16	17905.29	45740.25
GPU Utilization% (FIFO)	20498.89	5540.84	6413.51	14681.26	44198.14
GPU Utilization% (SJF)	13450.48	2776.40	5337.25	9874.39	42313.61
CPU Utilization% (FIFO)	3627.14	12201.24	15968.43	10646.34	31539.89
CPU Utilization% (SJF)	3362.36	8663.90	13568.19	8104.81	31469.66
GPU Memory (FIFO)	17076.69	11176.63	0	15786.05	30495.19
GPU Memory (SJF)	15543.98	12847.58	0	14954.14	29714.42

Table 18 Job distribution based on computing efficiency using individual parameters

	GTX1080	RTX2070	TitanX
GPU Memory Utilization% (FIFO)	0 (0%)	72 (56.25%)	56 (43.75%)
GPU Memory Utilization% (SJF)	0 (0%)	72 (56.25%)	56 (43.75%)
GPU Utilization% (FIFO)	76 (59.37%)	14 (10.94%)	38 (29.69%)
GPU Utilization% (SJF)	76 (59.37%)	14 (10.94%)	38 (29.69%)
CPU Utilization% (FIFO)	48 (37.5%)	14 (10.94%)	66 (51.56%)
CPU Utilization% (SJF)	48 (37.5%)	14 (10.94%)	66 (51.56%)
GPU Memory (FIFO)	100 (78.12%)	28 (21.88%)	0 (0%)
GPU Memory (SJF)	100 (78.12%)	28 (21.88%)	0 (0%)

4.6.2 Resource Consumption Prediction Removing One of Each Parameter

For Tab. 19, removing one of each resource consumption parameters from the model for scheduling jobs with FIFO and SJF were observed. It shows that when CPU Utilization% is removed, the jobs are only allocated on RTX2070 and TitanX. It also shows that removing GPU Memory Utilization% from the model increases the average JCT and Makespan of the entire proposed model by up to 2.25% and 5.99%, respectively.

Based on these results, it can be assumed that the GPU Memory Utilization% is not needed for the prediction model for scheduling, since removing one of the other parameters increases the JCT and Makespan of the scheduler. Also, it is notable that the scheduling based on GPU Memory Utilization% alone worsened the JCT and Makespan of the scheduler when using SJT.

In addition, it can be seen in Tab. 20 that CPU Utilization% highly influences the GPU allocation because

once it was removed there were 0 jobs allocated on GTX1080. On other cases, even if the other parameters were removed, the allocated jobs on GTX1080 remained consistent.

Table 19 Scheduling based on computing efficiency by removing one of each parameter (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
w/o GPU Memory Utilization% (FIFO)	3627.14	15216.02	10768.57	11901.85	33678.68
w/o GPU Memory Utilization% (SJF)	3285.68	9244.58	10534.79	7875.81	32699.8
w/o GPU Utilization% (FIFO)	3627.14	9212.53	22607.66	12630.06	41815.10
w/o GPU Utilization% (SJF)	3129.87	6839.89	15589.24	9056.68	41001.02
w/o CPU Utilization% (FIFO)	0	10933.61	27365.87	21460.53	47644.84
w/o CPU Utilization% (SJF)	0	6733.22	28965.25	20975.61	45447.84
w/o GPU Memory (FIFO)	3627.14	9586.68	21969.74	12546.96	41262.48
w/o GPU Memory (SJF)	3201.46	5358.39	15065.50	8307.65	39544.48

Table 20 Job Distribution based on computing efficiency by removing one of each parameter

	GTX1080	RTX2070	TitanX
w/o GPU Memory Utilization% (FIFO)	32 (25%)	84 (65.62%)	12 (9.38%)
w/o GPU Memory Utilization% (SJF)	32 (25%)	84 (65.62%)	12 (9.38%)
w/o GPU Utilization% (FIFO)	32 (25%)	50 (39.06%)	46 (35.94%)
w/o GPU Utilization% (SJF)	32 (25%)	50 (39.06%)	46 (35.94%)
w/o CPU Utilization% (FIFO)	0 (0%)	46 (35.94%)	82 (64.06%)
w/o CPU Utilization% (SJF)	0 (0%)	46 (35.94%)	82 (64.06%)
w/o GPU Memory (FIFO)	32 (25%)	50 (39.06%)	46 (35.94%)
w/o GPU Memory (SJF)	32 (25%)	50 (39.06%)	46 (35.94%)

Furthermore, it is also notable that by removing GPU Memory Utilization%, despite having only 12 jobs on TitanX, most of the jobs that has higher execution time was allocated to it and has comparable JCT compared to the RTX2070 that has 84 jobs allocated to it when using SJF Scheduling.

4.7 Scalability

Scalability of the GPU scheduling algorithm is tested to determine its behavior on larger scale systems. By increasing the number of available GPUs, the number of datasets that give out an efficiency score also increases. Furthermore, using non-neural network models as inputs and only having a simple graph as an input were analyzed to demonstrate that the algorithm does not specifically work only on neural network workloads, but it could also be designed to process non-neural network workloads.

The time complexity of the algorithm when using FIFO is shown in Fig. 8. It shows that the execution time grows linearly when GPU servers increase with the same number of jobs but when the jobs are also scaled up twice or four times the number of available GPU servers, the execution time of the algorithm grows based on the number of jobs x GPU servers. In addition, it is noticeable that there is a small bump in execution time on lower GPU servers, this is due to the overhead caused by loading up the Tensorflow library. Ideally, the algorithm only needs to boot up once and does not need to reload the Tensorflow library every time a new job appears in the queue.

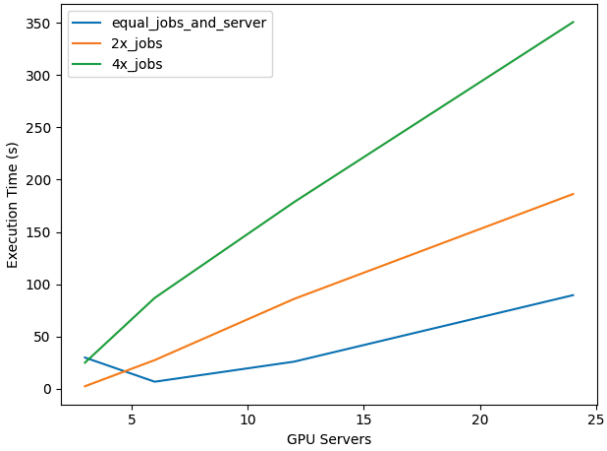

Figure 8 Execution time of the FIFO algorithm by scaling GPU servers and jobs

```
queue = get.Jobs()

for job in queue:                                                    #cost = O(n)
    inputs = convert_to_graph(jobs)

    for gpu in datasets:                                             #cost = O(m)
        resource_consumption_pred = prediction_model(inputs,gpu)
        comp_eff = sum(resource_consumption_pred[peak] / resource_consumption_pred[burst])
job.GPUAllocation(argmax(comp_eff))

if SJF:
    for each GPU:                                                    #cost = O(m)
        queue = sort.by(execution_time_pred,ascending)
```

Figure 9 Computational costs of the scheduling algorithm using big O notation

To further describe the behavior of the algorithm, Big O notation was utilized. As shown in Fig. 9, for O(n) where n stands for the number of jobs loaded into the queue, and O(m) where m stands for the number of GPU servers or datasets that are being used in the system. Based on the n and m parameters using the Big O notation, it is evaluated that the execution time for the FIFO Scheduling Algorithm will be O(n*m), which is determined by the number of jobs and number of GPU servers used in the system. For the SJF Scheduling Algorithm however, after allocation jobs are sorted for each GPU which adds another O(m) to the execution time.

This results in O(n*m) + O(m) for the SJF Algorithm. The expected average workload of the scheduling algorithm is 1 job and m number of GPU servers which leads into O(1*m) or O(m). Lastly, in the worst-case scenario there will be multiple jobs and multiple GPU servers to consider which

leads into a O(n*m) execution time. The summary of this data is shown in Tab. 21.

Table 21 Execution time for FIFO and SJF scheduling algorithms using big O notation

	Average	Worst-Case
(FIFO)	O(m)	O(n*m)
(SJF)	O(m) +O(m)	O(n*m) + O(m)

5 DISCUSSION AND CONCLUSION

5.1 Discussion

The experiments in this work used three GPU servers, specifically, TitanX, RTX2070, and GTX1080, solely due to availability. This work is not limited to these three GPUs only and can be implemented on other NVIDIA GPU types as well, since the framework of the system mostly runs on TensorFlow.

In scenarios where there are multiple GPU of the same type, a single dataset can be used to represent these GPU servers. For example, if there are three GTX1080 servers, a single dataset can be used for each of the GTX1080 servers since they all have similar capacity. However, since it is highly likely that due to them having a shared dataset, the scores will have the same or similar values, it is recommended that a priority system should be introduced when scheduling in systems with multiple GPU servers of the same type. Having the same GPU server also reduces the execution time of the algorithm since one dataset can represent multiple GPU servers in this scenario.

Limitations on the experiment setup used in this research is that the GPUs used in the work is limited to GTX1080, RTX2070 and TitanX and does not cover a wide range of GPU types which can help with analyzing the GPU performance for different training workloads that can contribute in creating a more generalized model used for scheduling jobs in cloud systems.

In addition, this work limited only with deep learning workloads. However, as long as a code can be represented into a graph, it can be trained and analyzed for a prediction model of different workloads not only limited to deep learning workloads. Adjusting the parameters and introducing new variables to incorporate a change in input.

There is also the lack of a general dataset for this type of application. Unlike other applications for images, videos, sentences, and the like, there is no fundamental dataset that can be used for resource consumption of deep learning workloads on different types of GPU. At the moment, this is very difficult since there's always new deep learning models that are being developed. However, the complexity of these models continues to increase and along with it the computing requirements to train such models in a reasonable amount of time also increases.

It is also worth noting that a lot of these deep learning models are also derived from a simpler form of its application. There are various deep learning models such as VGG and ResNet that fall under the CNN category where we can also infer that the architecture of these models have some similarity between them and when there is enough common application regarding the analysis of such architectures, a general dataset can be created that can be used as a reference data for future research regarding performance analysis and also job allocation of deep learning applications.

5.2 Using Non-Neural Networks as Input

When a different type of input is not a neural network, the scheduler will still be able to determine the GPU allocation given that the input is converted into a graph format using Tensorflow. By utilizing the tf.get_concrete_function() [21] to obtain the graph representation of a given function and the use of tf.io.write_graph() [22] to write the graph representation into a file for the scheduler to be able to process and convert it to its own adjacency and feature matrix that can be used against the prediction model to determine GPU allocation as shown in Figs. 10 and 11. The limitation for this approach is that the inputs must be in graph format, mainly using Tensorflow.

```
import tensorflow as tf

def simple_relu(x):
  if tf.greater(x, 0):
    return x
  else:
    return 0

tf_simple_relu = tf.function(simple_relu)
graph_def = tf_simple_relu.get_concrete_function(tf.constant(1)).graph.as_graph_def()
tf.io.write_graph(graph_def, '.', 'sample_graph' +'.pbtxt')
```

Figure 10 Obtaining the Graph representation of a function

```
node {
  name: "x"
  op: "Placeholder"
  attr {
    key: "_user_specified_name"
    value {
      s: "x"
    }
  }
  attr {
    key: "dtype"
    value {
      type: DT_INT32
    }
  }...
```

Figure 11 The output of the graph representation

5.3 Conclusion

The research was able to create multiple datasets, one for each machine used in this research, containing four resource consumption parameters (GPU Utilization, GPU Memory Utilization, CPU Utilization, and GPU Memory) and four prediction targets for each parameter (Average Burst Time, Average Idle Time, Average Peak Consumption, Execution Time) which totals up to 16 prediction metrics. By utilizing the datasets created, a prediction model that predicts both resource consumption parameters and the execution time of a given input was developed. To evaluate the performance of the proposed model against the previous work, both models are run on a FIFO and SJF scheduling mechanism.

The results show that GPU allocation based on computing efficiency, compared with the previous work, the average JCT is improved up to 40.9%. However, the makespan of the proposed model were still higher compared to the previous work. It has been discovered that removing one of the resource consumption parameters included in the proposed model, improves the overall JCT and Makespan by

an additional 2.25% and 5.99%, respectively. Due to these observations, future work is considered which involves removing GPU Memory Utilization% from the resource consumption parameters, and further analysis on Execution Time predictions, to seek improvements on both JCT and Makespan.

This work is also limited by using only Convolutional Neural Networks and some Generative models. Including more models from different applications and having a bigger dataset available for this application would help create a more generalized model for scheduling in the future.

Acknowledgements

This research was supported by Kumoh National Institute of Technology (2022-2023).

6 REFERENCES

[1] OpenAI (2023). GPT-4 Technical Report. *ArXiv*, abs/2303.08774. https://doi.org/10.48550/arXiv.2303.08774

[2] Jang, S. B. (2021). Deep neural network structure design for equipment failure prediction in smart factory. *Asia-pacific Journal of Convergent Research Interchange, 7*(12), 1-10. https://doi.org/10.47116/apjcri.2021.12.01

[3] Kim, D. (2018). Deep learning neural networks for automatic vehicle incident detection. *Asia-pacific Journal of Convergent Research Interchange, 4*(3), 107-117 https://doi.org/10.14257/apjcri.2018.09.11

[4] Kim, T. H. (2022).Video anomaly detection based on convolutional neural network. *Asia-pacific Journal of Convergent Research Interchange, 8*(11), 73-87. https://doi.org/10.47116/apjcri.2022.11.06

[5] Yu, G., Gao. Y, Golikov P. & Pekhimenko G. (2021). Habitat: A runtime-based computational performance predictor for deep neural network training. *Proceedings of the 2021 USENIX Annual Technical Conference*, USENIX ATC'21.

[6] Justus, D., Brennan, J., Bonner, S. & McGough, A. S. (2018). Predicting the computational cost of deep learning models. *2018 IEEE International Conference on Big Data (Big Data)*, 3873-3882. https://doi.org/10.1109/BigData.2018.8622396

[7] Viebke, A., Pllana, S., Memeti, S. & Kolodziej, J. (2019). Performance modelling of deep learning on Intel many integrated core architectures. *International Conference on High Performance Computing & Simulation (HPCS2019)*, 724-731. https://doi.org/10.1109/HPCS48598.2019.9188090

[8] Yang, G., Shin, C, Lee, J., Yoo, Y. & Yoo, C. (2022). Prediction of the resource consumption of distributed deep learning systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems, 6*, 1-25. https://doi.org/10.1145/3489048.3530962

[9] Pei, Z., Li, C., Qin, X., Chen, X. & Wei, G. (2019). Iteration time prediction for CNN in multi-GPU platform: Modeling and analysis. *IEEE Access* 1(1). https://doi.org/10.1109/ACCESS.2019.2916550

[10] Yang, G. (2022). Driple, https://github.com/gsyang33/Driple

[11] https://developer.nvidia.com/nvidia-system-management-interface

[12] Shin C., Yang G., Yoo Y., Lee J. & Yoo C. (2022). Xonar: Profiling-based job orderer for distributed deep learning. *IEEE 15th International Conference on Cloud Computing (CLOUD2022)*, 112-114. https://doi.org/10.1109/CLOUD55607.2022.00030

[13] Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A. & Zaharia, M. (2020). Heterogeneity-aware cluster scheduling policies for deep learning workloads. *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation,* 481-498.

[14] Duan, Y. &. Wu, J. (2021). Improving learning-based DAG scheduling by inserting deliberate idle slots. *IEEE Network, 35*(6), 133-139. https://doi.org/10.1109/MNET.001.2100231

[15] https://github.com/tensorflow/benchmarks/tree/master/scripts/tf_cnn_benchmarks, (2022)

[16] Shu, M. (2019). Deep learning for image classification on very small datasets using transfer learning.

[17] Weng, Q., Xiao, W., Yu, Y., Wang, W., Wang, C., He, J., Li, Y., Zhang, L., Lin, W. & Ding, Y. (2022). MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. *Symposium on Networked Systems Design and Implementation*. https://doi.org/10.21203/rs.3.rs-2266264/v1

[18] Lan, Z. Chen, M., Goodman, S., Gimpel, K., Sharma, P. & Soricut, R. (2019) ALBERT: A lite BERT for self-supervised learning of language representations. *ArXiv.* abs/1909.11942

[19] He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR2016)*, Las Vegas, NV, USA, 770-778. https://doi.org/10.1109/CVPR.2016.90.

[20] Zhu, R., Zhao, K., Yang, H., Lin, W., Zhou, C., Ai, B., Li, Y. & Zhou, J. (2019). AliGraph: A comprehensive graph neural network platform. *ArXiv*. abs/1902.08730 https://doi.org/10.48550/arXiv.1902.08730

[21] https://www.tensorflow.org/api_docs/python/tf/function

[22] https://www.tensorflow.org/api_docs/python/tf/io/write_graph

[23] Radford, A., Metz, L. & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *ArXiv*. abs/1511.06434 https://doi.org/10.48550/arXiv.1511.06434

[24] Mirza, M. & Osindero, S. (2014). Conditional Generative Adversarial Nets. *ArXiv*. abs/1411.1784. https://doi.org/10.48550/arXiv.1411.1784

[25] Chollet, F. (2020). https://github.com/keras-team/keras-io/blob/master/examples/generative/vae.py

[26] Poon, S. M. Y. (2022). Simulation of vehicle target detection based on embedded system. *Journal of Science and Engineering Research, 1*(2), 1-20. https://doi.org/10.56828/jser.2022.1.2.1

[27] Tang, J. C. K. (2022). Deep learning-based analysis of voiceprint data mining. *Journal of Science and Engineering Research*, 1(1), 1-14. https://doi.org/10.56828/jser.2022.1.1.1

[28] Liu, X., Xu, H. & He, L. (2015). A Formal Method of CPU Resources Scheduling in the Cloud Computing Environment. *International Journal of Grid and Distributed Computing, 8*(1), 133-144. https://doi.org/10.14257/ijgdc.2015.8.1.13

Authors' contacts:

Abuda Chad Ferrino, M.S. Student
Department of Computer AI Convergence Engineering,
Kumoh National Institute of Technology,
61 Daehak-ro, Gumi-si, Gyeongsangbuk-do, 39177, Republic of Korea
cfpabuda@gmail.com

Tae Young Choe, Professor
(Corresponding author)
Department of Computer AI Convergence Engineering,
Kumoh National Institute of Technology,
61 Daehak-ro, Gumi-si, Gyeongsangbuk-do, 39177, Republic of Korea
choety@gmail.com