https://doi.org/10.33472/AFJBS.6 9.2024.3769-3784

African Journal of Biological Sciences

Journal homepage: http://www.afjbs.com

Research Paper Open Access

Efficient GPU job allocation in Cloud using the predicted CPU and GPU usage properties

Abuda Chad Ferrino[1] and Tae Young Choe[2*]

[1]M.S. Student, Department of Computer AI Convergence Engineering, Kumoh National Institute of Technology, Korea

[2]Professor, Department of Computer AI Convergence Engineering, Kumoh National Institute of Technology, Korea

*Corresponding author: Tae Young Choe

Volume 6, Issue 9, 2024

Received: 09 March 2024

Accepted: 10 April 2024

Published: 20 May 2024
doi:10.33472/AFJBS.6.9.2024.3769-3784

Abstract

One objective of GPU scheduling in Cloud is to minimize the completion times of given deep learning models. Difficulties of GPU scheduling come from a diverse type of parameters including model architectures and GPU types. The previous GPU scheduling research had used a small set of 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 4.2%. on GPU Allocation based on Computing Efficiency compared to Driple, and the makespan is reduced up to 18.9% on GPU Allocation based on Execution time in comparison to Driple.

Keywords: *Deep Learning, Cloud Computing, GPU Job Scheduling, Convolutional Neural Network, 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], 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 on the Convolutional Neural Network (CNN) used in [4], the stacked 3x3 convolution kernels from VGG16 have been increased up to 7x7 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.[redacted]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. 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. Meanwhile, 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 totals up 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. Another work involves the use of a CPU [7] instead of a GPU for profiling CNNs of varying model architecture sizes. They mainly utilized forward propagation and backward propagation per image for their measurements to predict the execution time, which is also found in [9]. However, in [9], they used a multi-GPU approach and were limited by small-scale CNN model architectures. Lastly, in [8] they profiled an input task, as shown in Figure 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 describe the resource consumption input, namely 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.

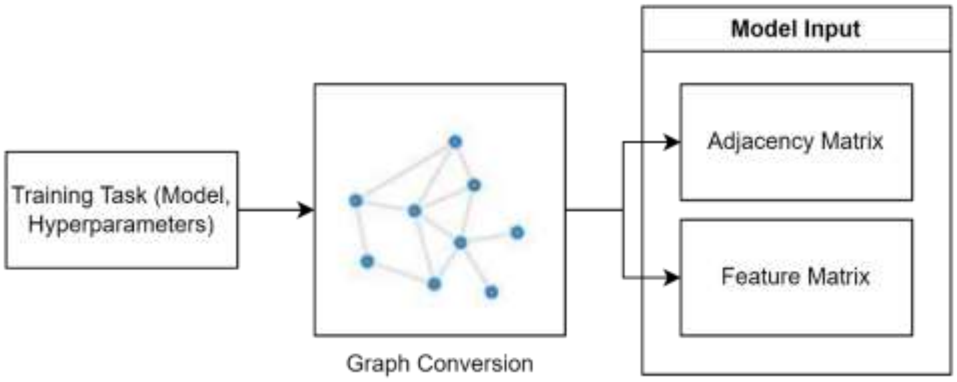


Figure 1. Input Task Conversion to Graph

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][17][18]. 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. 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 Figure 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 Util%, 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 Table 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-*

.

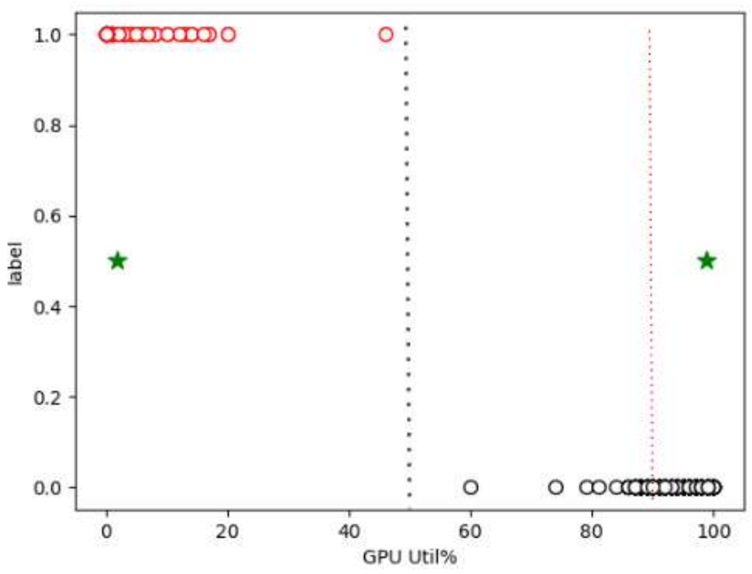


Figure 2. Output Parameters' boundary lines in a K-means Cluster

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.

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 Figure 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. The training workload consists of all the combinations of these settings, as shown in Table 3, which totals up to 384 training workloads for a given GPU.

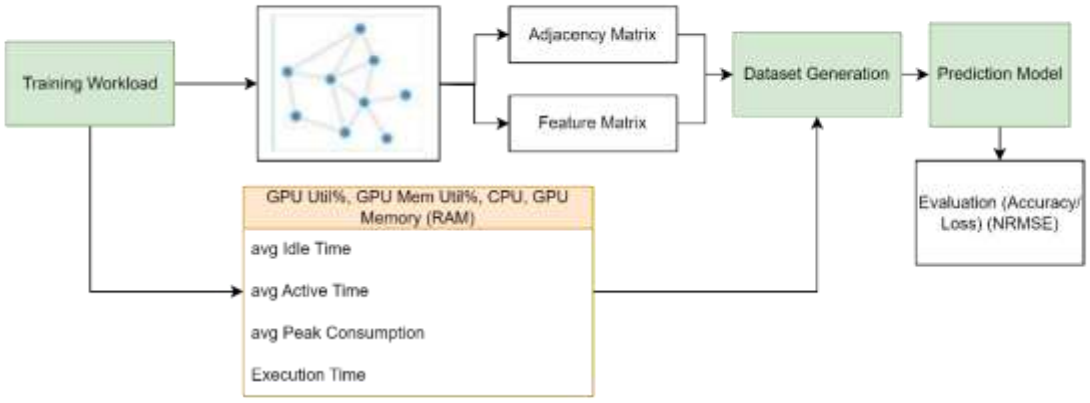


Figure 3. Job Profiler Workflow

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

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, having 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 Figure 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 Table 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 ŷ is the predicted value.

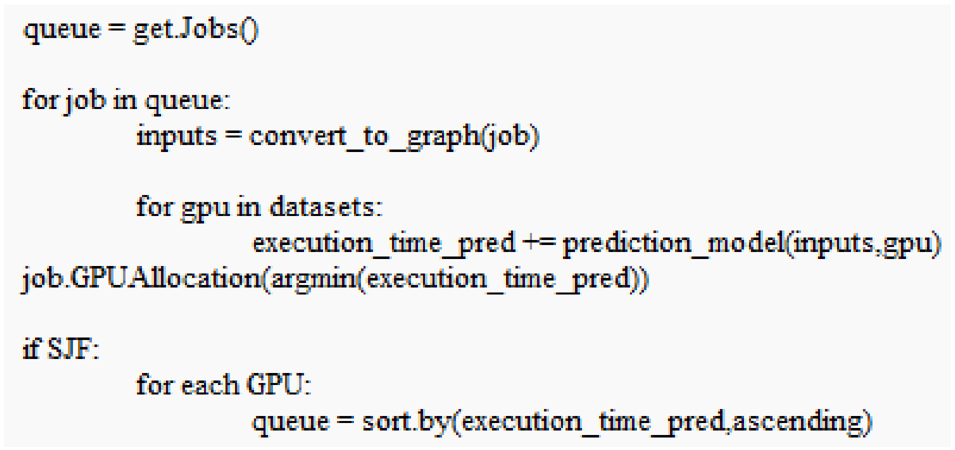
$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \qquad (1)$$

3.2. Job Scheduler

Once the prediction model is trained, the best model will be used by the job scheduler, as shown in Figure 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. 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 ($\hat{y}_{eff}$) by obtaining a summation of the quotient of the predicted average peak consumption rate ($\hat{y}_{peak}$) divided by the predicted average burst time($\hat{y}_{burst}$) for all resource consumption parameters.

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

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.



```
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.

```
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.

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 Table 4. Since the models trivial, alexnet, and resnet32 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 96 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

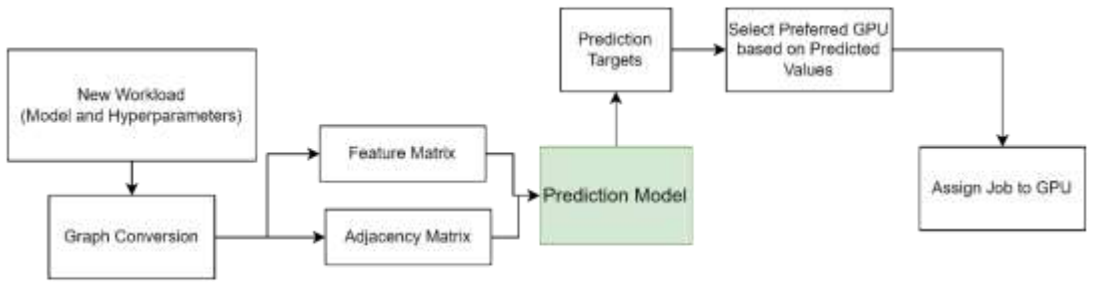


Figure 4. Scheduler Workflow

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 Table 5. The operating system used was Ubuntu 18.04, while using Tensorflow 2.5, and CUDA 11.2 to train a workload on two datasets, which are ImageNet and Cifar10. Training setting is specified by using a combination of different optimizers, batch sizes, epochs, and models as shown in Table 3, with the help of TensorFlow benchmark library [15]. For training the prediction model, recommended setup by [8] was used and executed on PyTorch 1.8.1

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

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 Figures 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.

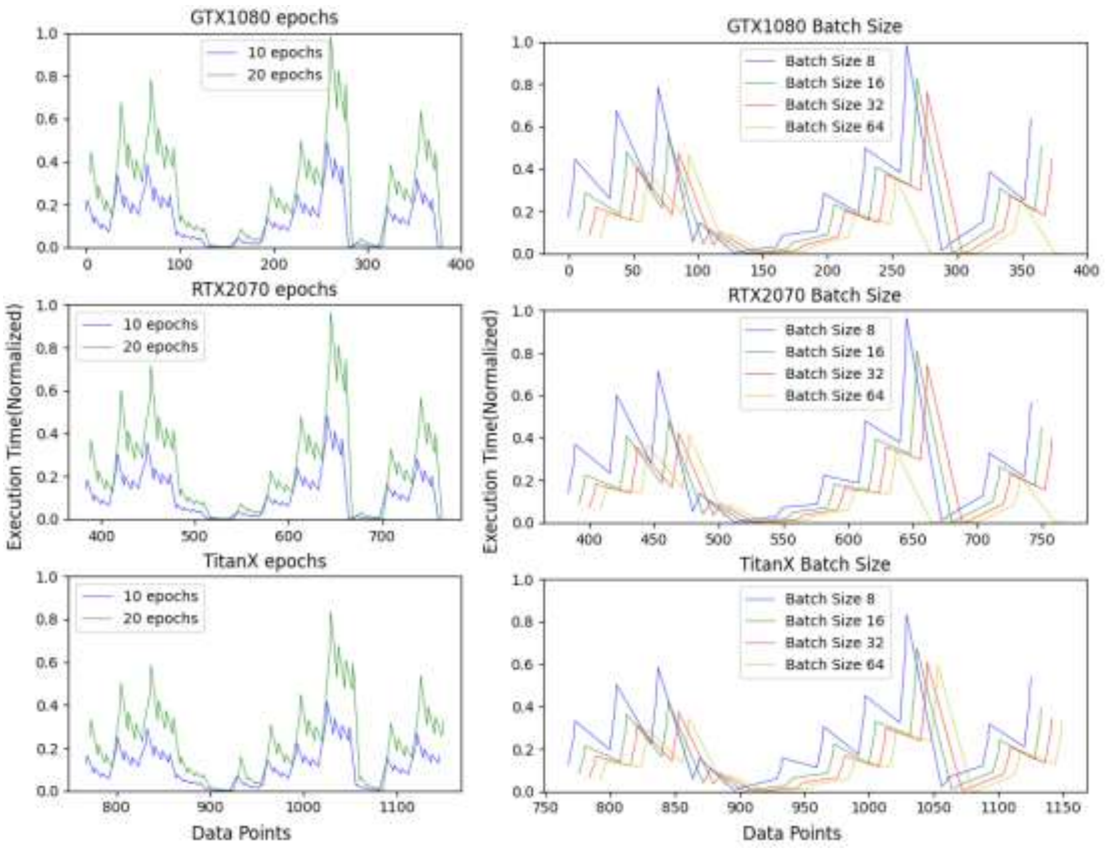


Figure 5. Execution Times based on Epochs and Batch Sizes among the 3 GPU Machines

For the optimizers, the difference in execution time is very minimal as shown in Figure 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 Figure 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 Figures 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 Figure 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.

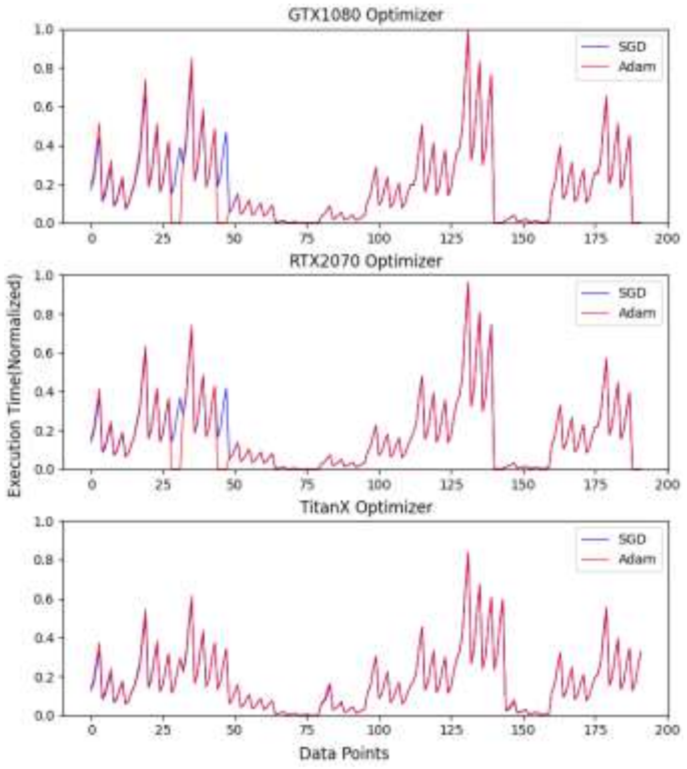


Figure 6. Execution Times based on Optimizers among the 3 GPU Machines

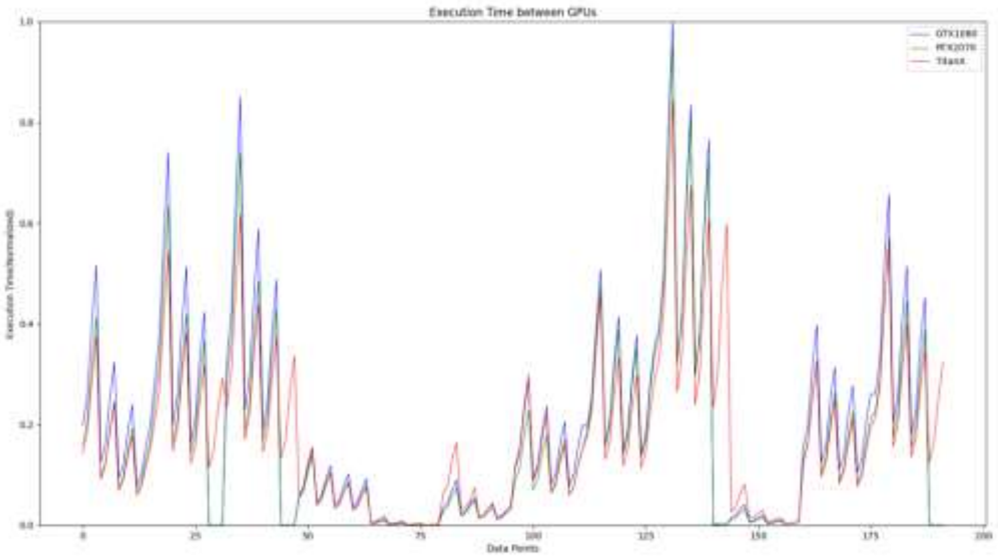


Figure 7. Execution Time Comparison between 3 GPU Machines

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, to evaluate the validation loss, training time, and epochs. The original model is based on the previous work [8] which uses GPU Utilization%, and GPU Memory Utilization% 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. Cases where the proposed model does not include Execution Time and removing one of the input parameters, such as GPU Memory, were also observed to compare the validation loss between these four setups. In Table 6, from the validation loss perspective, the original model 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. However, looking at the total loss across the three datasets, the proposed model w/o Execution Time performs the best among the four models tested. 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

Model	**Validation Loss**			**Evaluation**	
	GTX1080	**RTX2070**	**TitanX**	**Total Loss**	**Total Loss vs Original Model**
Original Model	0.0217	**0.0072**	0.0281	0.057	0
Proposed Model w/o Execution Time	0.0187	0.0098	0.0268	**0.0553**	-2.98245614
Proposed Model w/o GPU Memory	0.0236	0.012	0.0319	0.0675	+18.42105263
Proposed Model	**0.0171**	0.0239	**0.0199**	0.0609	+6.842105263

Considering the Validation Loss, it has been observed that the increase in input parameters increases the complexity of the model, while being advantageous on some datasets, are all seeing increases, among the sum of three different datasets in 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 Table 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. Evaluation of Execution Time and Computing Efficiency

By utilizing the proposed model with transfer learning as the prediction model, the jobs in Table 4 was loaded in as input to the prediction model to predict where each job should be allocated based on Execution Time and Computing Efficiency. For scheduling, execution time should be sufficient when trying to allocate jobs, but in the context of Deep Learning, it is also important to know how much resources would be needed for a job to perform optimally, hence computing efficiency ($\hat{y}_{eff}$) which takes the ratio of average peak consumption and burst time of a resource consumption parameter, was also explored. A simulation running 12 jobs on each GPU with the same parameters for both Execution Time and Computing Efficiency was tested on SJF Scheduling in two scenarios. First, the prediction model will predict the execution time and computing efficiency, and then the jobs will be sorted before or after the jobs are allocated to each of the available GPUs. In Table 8, the Average JCT and Makespan of the simulation show that for execution time, the Average JCT is lower if the jobs are sorted after allocation. Despite the makespan being a few seconds higher compared to sorting jobs before allocation, the difference is minimal. On the other hand, for the simulation of SJF in terms of efficiency, as observed in Table 9, the results are similar to Execution Time, however the difference between the average JCT is minimal, but the difference in makespan is substantial compared to the simulation in terms of Execution Time. Based on the results from these two simulations, sorting the jobs after allocation is preferable due to it being consistently lower in average JCT and having a better makespan in this setup.

Table 8. SJF Simulation for Execution Time Simulation (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
SJF sort job before allocation	6977.265	2282.544	2368.067	968.9897	**2090.189**
SJF sort job after allocation	2021.589	1421.938	7475.873	**909.95**	2095.376

Table 9. SJF Simulation for Efficiency (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
SJF sort job before allocation	1165.553	3350.324	7358.036	989.4928	3400.582
SJF sort job after allocation	4596.604	2503.162	4745.45	**987.1014**	**2148.096**

4.6. Performance Comparison based on Computing Efficiency

Since the previous work [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. Since the previous work does not have the Execution Time as its parameter, the comparison was based on the computing efficiency by using Equation 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. The GPU allocation mostly ignored the GTX1080 GPU for this setup, having 0 jobs allocated on this machine, and the rest of the jobs distributed on the other two machines for both FIFO and SJF, as observed in Tables 10 and 11. Still, it can be observed that while Average JCT is higher on FIFO, it is about 4.2% lower than Driple for SJF. However, looking at the makespan for each scheduling algorithm, it can be considered that Driple is superior, even though the proposed model is marginally higher by approximately 1.2% on SJF.

Table 10 FIFO based on Computing Efficiency (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
Driple	0	13814.65	14048.61	**13970.6**2	**30734.92**
Proposed Model	0	11135.98	19748.27	14903.86	35929.98

Table 11. SJF based on Computing Efficiency (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
Driple	0	13182.03	5737.863	9459.945	**29827.69**
Proposed Model	0	5957.097	14534.12	**9060.584**	30182.41

4.7. GPU Allocation based on Execution Time

This time, the proposed model based on execution time prediction was observed for FIFO and SJF algorithm, the predicted execution time with the minimum value was the basis for GPU allocation. As shown in Table 12, the scheduling based on the execution time prediction of the proposed model increases by approximately 19.8% when using SJF scheduling compared to FIFO. However, the makespan decreases by approximately 12.66% when using SJF scheduling. Comparing this with the prediction results based on computing efficiency, the makespan is reduced by up to 18.9%.

Table 12. Scheduling based on Execution Time Prediction (seconds)

	GTX1080	RTX2070	TitanX	Average JCT	Makespan
Proposed Model (FIFO)	1137.847	10285.88	11487.41	9367.663	27694.29
Proposed Model (SJF)	1170.366	12684.82	12580.51	11225.95	**24190.34**

As seen in the GPU allocation based on Execution Time and Computing Efficiency, the scheduler performed better on makespan when based on Execution Time, and on average JCT, when based on computing efficiency. Regardless, GPU Allocation based on computing efficiency ignores one of the machines, so even if the JCT was ideal there is still room for improvement. While the GPU Allocation based on Execution Time improves the makespan and uses all the machines, the JCT higher compared to the allocation based on Execution Time. Hence, improvements in future works, such as combining both the efficiency and execution time to improve both JCT and Makespan are considered.

5. Conclusion

On multiple tests on the hyperparameter settings used in this research, it was found that epochs are the major factor when it comes to execution time, with the batch size also having a big impact if the batch size is smaller. With a growing batch size, the reduction in execution time becomes less apparent. For smaller datasets, it is concluded that transfer learning performs way better than models without transfer learning, improving the model accuracy by 5.09%. By simulating job sorting for SJF before allocation and after allocation, it was found that sorting jobs after allocation gives a 6.09% boost in average JCT, while makespan can be improved by up to 36.83%. Using this information, the scheduling mechanism was tested based on execution time and computing efficiency using FIFO and SJF (sorting after job allocation). The results show that GPU allocation based on computing efficiency, compared with the previous work, the average JCT is improved up to 4.2%. While on the case of GPU allocation based on execution time, the makespan was improved up to 18.9% compared to the previous work. However, there were still cases where the GPU allocation based on Execution Time increased the average JCT, and the allocation based on computing efficiency, did not distribute the jobs to all the available machines. Due to these observations, future work is considered which involves combining both the GPU allocation based on Computing Efficiency and GPU Allocation based on Execution Time, to seek improvements on both JCT and Makespan. This work is also limited by using only Convolutional Neural Networks. 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.

6. Acknowledgements

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

References

[1] OpenAI (2023). GPT-4 Technical Report. *ArXiv*, abs/2303.08774.
doi: 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, FuCoS, ISSN : 2508-9080 (Print); 2671-5325 (Online), Vol.7, No.12, 1-10.
doi: http://dx.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.* SoCoRI, ISSN : 2508-9080 (Print); 2671-5325 (Online), Vol.4, No.3, pp. 107-117
doi: http://dx.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*, FuCoS, ISSN : 2508-9080 (Print); 2671-5325 (Online), Vol.8, No.11, pp. 73-87
doi: http://dx.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.
doi: 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. *2019 International Conference on High Performance Computing & Simulation (HPCS)*, 724-731.
doi: 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.
doi: https://doi.org/10.1145/3530895

[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. PP. 1-1.
doi: 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. *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pp. 112-114
doi: https://doi.org/10.1109/CLOUD55607.2022.00030

[13] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, M. Zaharia (2020). Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation,* PP. 481-498

[14] Y. Duan and J. Wu (2021). Improving Learning-Based DAG Scheduling by Inserting Deliberate Idle Slots. *IEEE Network*, vol. 35, no. 6, pp. 133-139
doi: 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] Tanya Street and Jugal Simelane. (2019). Research on Cloud Service Provider Recommendation Model Based on User Preference. International Journal of Smart Business and Technology, vol.7, no.1, May, (2019), Global Vision Press, pp:1-16,10.21742/IJSBT.2019.7.1.01

[18] Anitha R and C Vidyaraj (2019). An Adaptive Computational Model for Threshold Based VM Migration and Job Scheduling in Cloud. International Journal of Future Generation Communication and Networking, vol.12, no.2, pp. 1-10). Doi: 10.33832/ijfgcn.2019.12.2.01.