Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Research Project

Cluster Scheduling in Data Centers

Piyush Krishan Bajaj

Course of Study:	INFOTECH
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Dr. Kawsar Haghshenas
Commenced:	April 25, 2022
Completed:	October 26, 2022

Abstract

Cloud Computing allows on demand access to various resources over the internet. Deep Learning and Machine Learning are an emerging trend in the recent decade. Deep Learning training (DLT) jobs are compute intensive and time consuming. These DLT jobs often require the usage of high end devices such as GPUs, TPUs, FPGAs and ASICs in order to complete in a timely fashion. These high-end devices cost nearly 10x and are not abundantly available compared to normal CPUs. Cloud providers setup such resources and make them available to the different users including startups and established companies. These resources setup on the cloud, can be accessed based on monthly subscription or pay-as-per-use model. sharing them amongst users is a great way of saving cost and at the same time allowing many users access to them. These resources need to be shared amongst many users at the same time. Hence, scheduling plays a vital role in such clusters to manage and distribute the heterogeneous resources efficiently among the users and their tasks. There are many existing schedulers and policies which try to optimize one or the other objective such as job completion time, makespan, etc. DLT jobs have different characteristics as compared to normal jobs and hence need some adjustments over the existing scheduling policies. Gavel, which is a scheduler specialized for DLT jobs incorporates various scheduling policies such as FIFO, AlloX, Themis to efficiently distribute DLT jobs. Each individual policy in gavel is converted into an optimization problem and returns a target allocation of jobs on different resources to achieve specific objective such as minimum makespan, average job completion time, etc. In this project, we first analyse the throughput, given as steps per second, of various neural network models based on their batch sizes and the resources they run on. This analysis provides us the details about the impact of various factors such as inter-arrival time between jobs, model weights, model architecture and resource architecture on the running time of model on the heterogeneous resources. Next, we study the cluster utilization of various policies in the gavel simulator. This task helps us in understanding the impact of objective functions on the usage of heterogeneous cluster resources. Finally, we implement a new cluster scheduling policy, named *Shared Cloud Cost Fairness*, in gavel simulator which aims to optimize the ratio of shared cost to the individual cost of running DLT jobs in the cloud cluster. We then run simulated experiments to find that the new policy drastically improves the cost spent on the cloud resources while ensuring the same levels of cluster utilization compared to the other policies implemented in gavel. The code for this project is available on Github at [22m].

Contents

1 Introduction **13**

2 Background Information **15**
2.1 Cloud Computing and Cluster 15
2.2 Scheduling 15
2.3 Deep Learning 15
2.4 Gavel 16

3 State of the Art **17**
3.1 Overview of Gavel Scheduler 17
3.2 Policies implemented in gavel 18

4 Research Work **23**
4.1 Throughput Analysis 23
4.2 Cluster Utilization Analysis 26
4.3 Shared Cost Fairness Metric 29
4.4 Implementation in Gavel 30

5 Algorithm **33**

6 Evaluation **35**
6.1 Experimental Setup 35
6.2 End-to-End Results on Simulated Cluster 36

7 Conclusion **41**

8 Future Work **43**

Bibliography **45**

List of Figures

3.1 Gavel overview. 17
3.2 AlloX Scheduler Algorithm 19
3.3 Themis Pseudocode 20
3.4 Themis Scheduler 21

4.1 Throughput vs Batch Size for LSTM for different scales on P100 GPU 23
4.2 Throughput vs Batch Size for Resnet-18 for different scales on K80 GPU 24
4.3 Throughput of different models for scale 1 on K80 GPU. The batch size used is the one that provides best throughput 24
4.4 Throughput of different models for scale 2 on K80 GPU. The batch size used is the one that provides best throughput. This is the unconsolidated configuration, where the GPU are present on different servers 25
4.5 Throughput of Autoencoder on P100 GPU in case of unconsolidated configuration 25
4.6 Throughput of all models for their best batch size on a P100 GPU for scale 1 . . . 26
4.7 Utilization vs Lambda (Cluster Specification: v100=10, p100=5, k80=5) 27
4.8 Average Completion Time vs Lambda (Cluster Specification: v100=10, p100=5, k80=5) 27
4.9 Utilization vs Total Jobs (Cluster Specification: v100=5, p100=10, k80=5) 28
4.10 Average Completion Time vs Total Jobs (Cluster Specification: v100=5, p100=10, k80=5) 29

6.1 Job Configurations in Gavel 36
6.2 Cluster Utilization vs Lambda (Cluster Configuration:v100=10, p100=5, k80=5) . 37
6.3 Cluster Utilization vs Total Jobs (Cluster Configuration:v100=10, p100=5, k80=5) 37
6.4 Average Completion Time vs Lambda (Cluster Configuration:v100=10, p100=5, k80=5) 38
6.5 Average Completion Time vs Total Jobs (Cluster Configuration:v100=10, p100=5, k80=5) 38
6.6 Total Cost vs Total Jobs for all policies 39
6.7 Total Cost vs Total Jobs for Shared Cloud Cost Fairness 39

List of Tables

4.1 Cluster utilization and Average completion time for each Policy in Continuous trace. 27
4.2 Cluster utilization and Average completion time for each Policy in Static trace. . . 29

List of Algorithms

5.1 Shared Cloud Cost Fairness 33

1 Introduction

Cloud computing enables on-demand access to the computing resources in a remote data center over the internet. The computing resources at the data center, such as physical servers, virtual servers, development tools, and more are managed by a cloud service provider (CSP). The CSP provides such a cloud cluster on a monthly subscription basis or on the basis of usage of individual nodes. Clusters have seen a substantial rise in adoption in the last decade. Cluster-based architectures are being leveraged by Startups and tech-giants alike, to deploy and manage their applications in the cloud [22l]. Deep learning and machine learning are showing significant impact on many businesses across various domains such as image recognition, object detection and machine translation. However, these approaches require a lot of time and computing resources which are provided to an extent by the use of various specialized accelerators such as GPUs, TPUs, FPGAs, and custom ASICs. The trend now-a-days is to have such resources accessible over the internet via cloud. Deep learning and machine learning models are trained in parallel on a number of such accelerator resources consolidated into a cluster. Schedulers are used on these clusters to perform scheduling decisions based on the information about the load, the queues on compute nodes and available resources. Hence, cluster schedulers are key components in data centers. In the upcoming decade, Deep Neural Network (DNN) training workloads are a growing cluster workloads [17]. These workloads posses unique attributes of heterogeneity and placement sensitivity along with long-running and iterative tasks whose performance is dependent on the task's relative placement. As a result there is a lot of research to find new and better suited cluster scheduling policies that can utilize these features of the DNN workloads to achieve certain objectives. Some of the objectives proposed in the recent papers include improving efficiency and performance [XBR+18], fairness [MBS+19], and application quality [ZSOF17]. Gavel, a heterogeneity-aware scheduling policy was proposed by Deepak et al [NSK+20]. Gavel expresses a number of cluster scheduling policies such as Allox [LSCL20], Themis [MBS+19], Gandiva [XBR+18], Tiresias [GCS+19], and FIFO as optimization problem and then systematically transforms these problems into heterogeneity-aware versions using an abstraction called effective throughput. Gavel [NSK+20] uses preemptive round-based scheduling mechanism to ensure jobs receive resources as per the computed target allocations. This has an impact on cluster utilization due to the scheduling policies and at the same time improves the average job completion time, thereby reducing the cost of using cluster resources. Gavel also achieves the objective of the scheduling policy it uses such as Allox [LSCL20], Gandiva [XBR+18], etc. Gavel is publicly available with its GitHub repository [22n] containing the implementation of scheduling policies mentioned above.

This project performs the following tasks :

- Analysis of throughput with regards to the application type and parameters of different neural network models.
- Analysing the cluster utilization as given by Gavel simulator based on the usage of different scheduling policies.

- Implementing a new scheduling policy in the Gavel simulator to minimize shared cost resulting in improvement of cluster utilization.

2 Background Information

We now look at the various topics that are relevant for the understanding of this project.

2.1 Cloud Computing and Cluster

A cluster is a group of two or more computers, or nodes that run in parallel. As a result, the workloads can be distributed among the nodes in the cluster to leverage the combined memory and processing power. Data center contains IT Infrastructure for building, running and delivering applications and services. They also store and manage the data associated with these applications and services. Cloud computing enables on-demand access to the computing resources in a remote data center over the internet. The computing resources at the data center, such as physical servers, virtual servers, development tools, and more are managed by a CSP. The CSP provides such a cloud cluster on a monthly subscription basis or on the basis of usage of individual nodes. Clusters have seen a substantial rise in adoption in the last decade. Cluster-based architectures are being leveraged by Startups and tech-giants alike, to deploy and manage their applications in the cloud [22l].

2.2 Scheduling

Scheduling is pivotal in the cloud computing systems. As a result we need efficient schedulers, which is a challenging task due to multi-dimensionality of resource demands, heterogeneity of jobs, diversity of computing resources, and fairness between multiple tenants sharing the cluster. There are many schedulers available which work on different scheduling policies aiming to achieve a certain objective, such as early finish time, fairness amongst users, first come first serve and many more. There are many challenges faced by existing cluster scheduling systems such as achieving a tradeoff between multiple conflicting objectives, finding the balance between jobs' requirements, scaling to the new operational demands, and choosing the appropriate scheduling architecture [KH22].

2.3 Deep Learning

Deep Learning is type of representation learning that automatically infers from raw data to accomplish tasks such as image classification, language translation, language modelling, etc. Deep Learning is represented by a deep neural network model with millions of weights carefully arranged in layers. Deep Learning training is used to learn these model weights. Training operates on a subset of data known as a mini-batch and computes a score based on numerical computations using the model weights. This is known as a forward pass. An objective function that measures an error

between computed scores and desired scores is used to calculate gradient for each weight and then update the weight via a backward pass over the model to decrease the error. The backward and the forward pass, both typically involve billions of floating point operations. Each forward-backward pass over the model is a mini-batch iteration. Millions of such iterations are required to be performed on large data sets in order to achieve high accuracy of the model. Deep Learning has become an increasingly popular trend over the last few years [LBH15] and has significant potential to impact businesses. It has become a vital and a growing workload in data centers. Deep Learning is compute intensive and in order to perform millions of floating point operations it is reliant on powerful and expensive GPUs. GPU VM costs nearly 10x compared to a regular VM. Large companies and cloud operators manage clusters of tens of thousands of GPUs and rely on efficient cluster schedulers to ensure high utilization of GPUs. Traditional schedulers such as Kubernetes [BGO+16], YARN [VMD+13] treat the deep learning training job as a normal job and allocates exclusive access to a set of GPUs to the job until its completion.

2.4 Gavel

In order to train deep learning models, various accelerators such as GPUs, TPUs, FPGAs and custom ASICs which exhibit heterogeneous behavior across model architecture are being deployed. Today, users must choose from a wide variety of accelerators to train their Deep Neural Network models. GPU cluster schedulers such as Themis [MBS+19], Tiresias [GCS+19], AlloX [LSCL20], and Gandiva [XBR+18] are used to arbitrate these expensive resources and thus need to infer on the allocation of diverse resources to many users based on complex cluster-wide scheduling policies in order to achieve objectives such as fairness or minimum makespan. These schedulers unfortunately do not consider performance heterogeneity. Gavel, which is a heterogeneity-aware scheduler generalizes a wide range of existing scheduling policies into an optimization problem. This make the policy easier to optimize for the objectives in a heterogeneity-aware way, while also being sensitive to performance optimizations like space sharing. In order to ensure jobs receive an ideal allocation, Gavel uses round based scheduling mechanism for the given scheduling policy. Gavel helps to improve end objectives such as average completion time and makespan while sustaining higher input load in a heterogeneous cluster [NSK+20].

3 State of the Art

We now understand the gavel scheduler and some of the policies that it implements.

3.1 Overview of Gavel Scheduler

The cluster scheduler, Gavel, integrates heterogeneity in hardware accelerators and DNN workloads for DNN training in order to universalize a wide range of existing policies such as fair sharing / least attained service [GCS+19], FIFO, minimum makespan, minimum cost subject to SLOs, finish-time fairness [MBS+19], shortest job first, and hierarchical policies [Nie17; ZBS+10]. Gavel works on the observation that the scheduling policies can be represented as optimization problems whose objective is a function of the jobs' achieved throughputs. Consider the example of least attained service, the objective is equivalent to maximizing the minimum scaled throughput among the jobs. Similarly we can deduce optimization problems for other policies as well. Gavel transforms the problem to make it heterogeneity-, colocation- and placement-aware and these problems can be efficiently executed on clusters with hundreds of GPUs and jobs. Most of these optimization problems can be solved with the help of one or more linear program solvers such as ECOS, SCS and GUROBI. Pre-emptive round based scheduling mechanism is used in order to ensure jobs receive the computed target allocation. Gavel specifies an API between the application and scheduler, allowing jobs written in PyTorch [PGM+19] and Tensorflow [ABC+16] to shift between heterogeneous resources with minimal code changes. In order to estimate the performance measurements of co-located jobs, Gavel uses a mechanism similar to Quasar [DK]. The performance measurements are provided as input to the Gavel's policies. Gavel enhances policy objectives by up to 1.4x on a small physical cluster, whereas on a large simulated cluster, it increases the maximum cluster load, while improving objectives such as average job completion time by 3.5×, makespan by 2.5×, and cost by 1.4× [NSK+20].

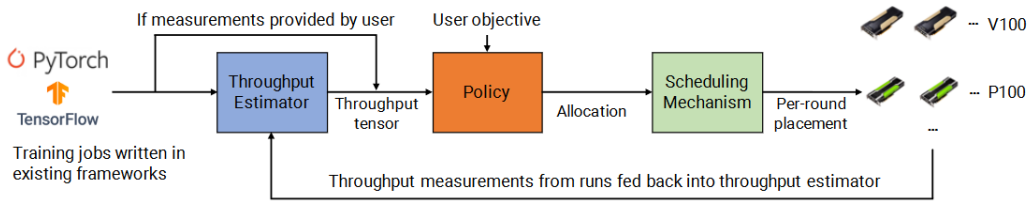

Figure 3.1: Gavel overview.

3.2 Policies implemented in gavel

3.2.1 First Come First Serve

First Come First Serve (FCFS) is also known as the First In First Out (FIFO). It is a non pre-emptive scheduling policy. It works by adding all the jobs to a queue in the order in which they arrive. The arrival times can be considered as the priority order for the jobs. One job is picked from the queue and is then assigned to the resource amongst the remaining ones which provides the best throughput. Throughput is defined as steps per second for the individual job. Once all the resources are assigned, the remaining jobs then wait in the queue till one or more resources are made available again. Since FCFS is a non pre-emptive algorithm, it does not kill the running jobs. Gavel reassigns the same allocation for the already running jobs in each round till the running job completes its execution. The objective function can be written as:

$$Maximize_x \sum_m \frac{throughput\,(m, X)}{throughput\left(m, X^{fastest}\right)}(M - m)$$

3.2.2 AlloX [LSCL20]

AlloX is a policy that optimizes performance while providing fairness among users in the shared cluster. AlloX transforms the scheduling problem into a min-cost bipartite matching problem and provides dynamic fair allocation over time. It assumes that each job uses either an entire GPU or an entire CPU. For arriving jobs we consider jobs having multiple configurations as GPU, CPU, etc., Then we solve the min-cost bipartite graph problem. Specifically, the algorithm consists of three steps: (i) generate input for the min-cost bipartite matching problem based on job information; (ii) solve the matching problem to obtain a solution; (iii) convert the solution to a feasible scheduling and placement.

A job scheduled as the k-th last one contributes k times its processing time to the total job completion time. The job processing time can be represented by the processing time matrix P where each row contains processing time on CPU and GPU. Based on the processing time matrix P, we can generate the cost matrix Q of size n × (nm) as $Q = [\ P\ 2P\ 3P\ ....\ nP\ ]$. Given the matrix Q, we can formulate the problem into a min-cost bipartite matching problem. This min-cost bipartite matching problem can be solved in polynomial time with standard network flow algorithms or Hungarian method [Kuh55; Orl93]. Each edge picked by the matching algorithm correspond to the scheduling and placement of a job. This policy does not consider preemption. For handling online arrivals, We assume that the scheduler has no information about future arrivals. The major difference with arrivals over time is that when we generate a new schedule, some machines are occupied so new jobs need to wait until current jobs finish. We use both the arrival time of new jobs and the available time of machines (defined shortly) to adjust the cost matrix Q for the matching problem. We define delay matrix $D(j,i) = \omega(i) - a(j)$, where $a(j)$ is the arrival time of job j, and $\omega(i)$ is the earliest available time of machine i when it finishes its currently allocated job(s). If machine i is idle, then $\omega(i) = T$, where T is the current time. The cost matrix Q is calculated by the following:

$$Q = [\ P\ 2P\ ...\ nP\ ] + [\ D\ D\ ...\ D\ ]$$

Algorithm AlloX Scheduler

function SCHEDULENEXTJOB(available machine i)
  Update users' progress and get the set of users A_α consisted of $\lceil \alpha n \rceil$ users with the lowest progress.
  for all job j in the waiting queue from A_α **do**
    Add processing time of job j to matrix P
  end for
  Generate the delay matrix D and further the cost matrix Q;
  Solve the min-cost matching problem defined by Q to get the matching matrix M
  for $k = J : 1$ **do** ▷ J is the total # of jobs in A_α
    for $w = 1 : J$ **do**
      if $M(w, m(k-1)+i) = 1$ **then** ▷ w is first job scheduled on machine i
        schedule job w to machine i
        Update available time ω_i and users' progress
        return job w
      end if
    end for
  end for
  return null
end function

Figure 3.2: AlloX Scheduler Algorithm

, where P is processing time matrix. Delay matrix D is used to handle the non-preemption constraint.

3.2.3 Themis [MBS+19]

Themis is a policy that focuses on finish-time fairness. In the short term, this policy trades off efficiency for fairness. It works on the principle that sharing a cluster becomes attractive only if the sharing incentive is appropriate, i.e., sharing is as good as running individually. So, if there are N users sharing a cluster with C resources, then every user's performance should not be worse than using a private cluster of of size $\frac{C}{N}$. Themis takes into account the fact that placement of DL jobs in the cluster can significantly impact their speed. Themis uses a metric called as the finish-time fairness, $\rho = \frac{T_{sh}}{T_{id}}$. T_{id} is the independent finish-time and T_{sh} is the shared finish-time. Sharing incentive is attained if $\rho \leq 1$. The independent and the shared finish-time varies with time due to

$\vec{G}$	[0,0]	[0,1] = [1,0]	[1,1]
ρ	ρ_{old}	$\frac{200}{400} = \frac{1}{2}$	$\frac{100}{400} = \frac{1}{4}$

Table .: Example table of bids sent from apps to the scheduler

Pseudocode . Finish-Time Fair Policy

Applications $\{A_i\}$ ▷ set of apps
Bids $\{\rho_i(.)\}$ ▷ valuation function for each app i
Resources $\vec{R}$ ▷ resource set available for auction
Resource Allocations $\{\vec{G}_i\}$ ▷ resource allocation for each app i

procedure AUCTION($\{A_i\}$, $\{\rho_i(.)\}$, $\vec{R}$)
  $\vec{G}_{i,pf} = \arg\max \prod_i 1/\rho_i(\vec{G}_i)$ ▷ proportional fair (pf) allocation per app i
  $\vec{G}^{-i}_{j,pf} = \arg\max \prod_{j!=i} 1/\rho_j(\vec{G}_j)$ ▷ pf allocation per app j without app i
  $c_i = \frac{\prod_{j!=i} 1/\rho_j(\vec{G}_{j,pf})}{\prod_{j!=i} 1/\rho_j(\vec{G}^{-i}_{j,pf})}$
  $\vec{G}_i = c_i * \vec{G}_{i,pf}$ ▷ allocation per app i
  $\vec{L} = \sum_i 1 - c_i * \vec{G}_{i,pf}$ ▷ aggregate leftover resource
  return $\{\vec{G}_i\}$, $\vec{L}$
end procedure
procedure ROUNDBYROUNDAUCTIONS($\{A_i\}$, $\{\rho_i(.)\}$)
  while True **do**
    ONRESOURCEAVAILABLEEVENT $\vec{R'}$:
    $\{A_i^{sort}\}$ = SORT($\{A_i\}$) on $\rho_i^{current}$
    $\{A_i^{filter}\}$ = get top $1-f$ fraction of apps from $\{A_{sort}\}$
    $\{\rho_i^{filter}(.)\}$ = get updated $\rho(.)$ from apps in $\{A_i^{filter}\}$
    $\{\vec{G_i^{filter}}\}$, $\vec{L}$ = AUCTION($\{A_i^{filter}\}$, $\{\rho_i^{filter}(.)\}$, $\vec{R'}$)
    $\{A_i^{unfilter}\} = \{A_i\} - \{A_i^{filter}\}$
    allocate $\vec{L}$ to $\{A_i^{unfilter}\}$ at random
  end while
end procedure

Figure 3.3: Themis Pseudocode

CPU usage, remaining workload, etc. Hence, we calculate the value of ρ for each job and then try to solve the optimization problem for the value of ρ which is the worst out of all the jobs. T_{id} remains nearly constant for all jobs, however, the value of T_{sh} varies based on the allocation. The objective function can be written as follows:

$$Minimize_x \max_j \frac{T_{sh}}{T_{id}}$$

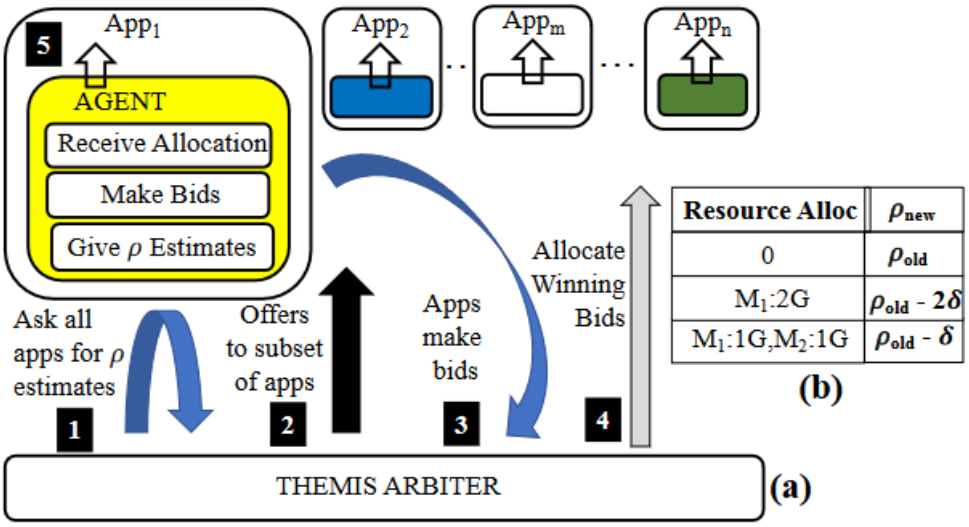

Figure : THEMIS Design. (a) Sequence of events in THEMIS - starts with a resource available event and ends with resource allocations. (b) Shows a typical bid valuation table an App submits to ARBITER. Each row has a subset of the complete resource allocation and the improved value of ρ_{new}.

Figure 3.4: Themis Scheduler

4 Research Work

We now present the work done as part of the Research Project.

4.1 Throughput Analysis

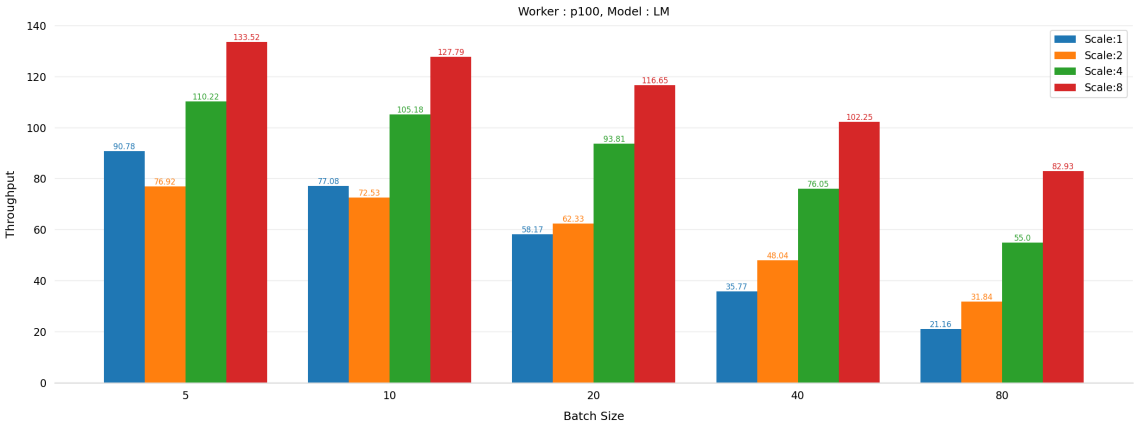

Figure 4.1: Throughput vs Batch Size for LSTM for different scales on P100 GPU

In this section we present the work done in the part of throughput analysis task. We used the file simulation-throughputs.json to study the throughput of various DLT jobs provided in the figure 6.1. The units of throughput are iterations or steps per second, where each iteration or step is one forward backward pass in the given model on a given mini-batch size. We analyzed the throughputs of different DLT jobs/model with its batch size and scale on a particular GPU resource known as worker. We try to provide some insights into the trends observed from these graphs based on throughput analysis. These insights are more of an hypothesis, since, we could only do our experiments on simulated cluster. We show some of the graphs generated from this analysis, for more graphs refer to the analysis folder at [22m].

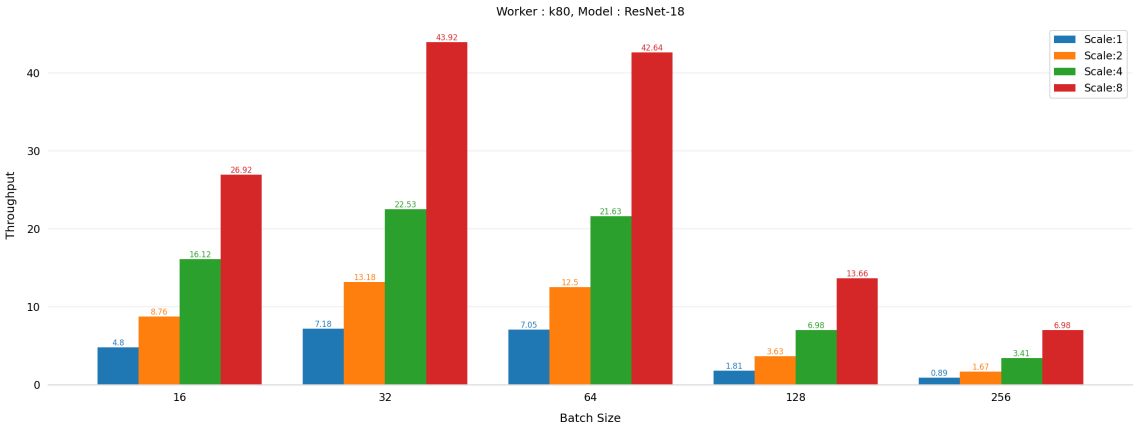

Figure 4.2: Throughput vs Batch Size for Resnet-18 for different scales on K80 GPU

The graphs in the figure 4.2, 4.3, 4.4, **??**, 4.1, 4.5, 4.6 and **??** represent the throughputs for various scales, batch sizes, models and workers. The term worker refer to the a single GPU resource, e.g., 1 K80 machine. The term scale in this case refers to the required number of workers the DLT job runs on simultaneously in order to function properly. Batch-size is the mini-batch size of the Deep Learning models, which involve various CNN and RNN models such as [HZRS15; kua22][], Resnet-50 [22a; HZRS15], Auto-encoder [Mou18], Language Modeling [22b] etc. There are two configurations for workers available namely consolidated, where all the workers are present on the same server and unconsolidated, where all the workers are present on different servers.

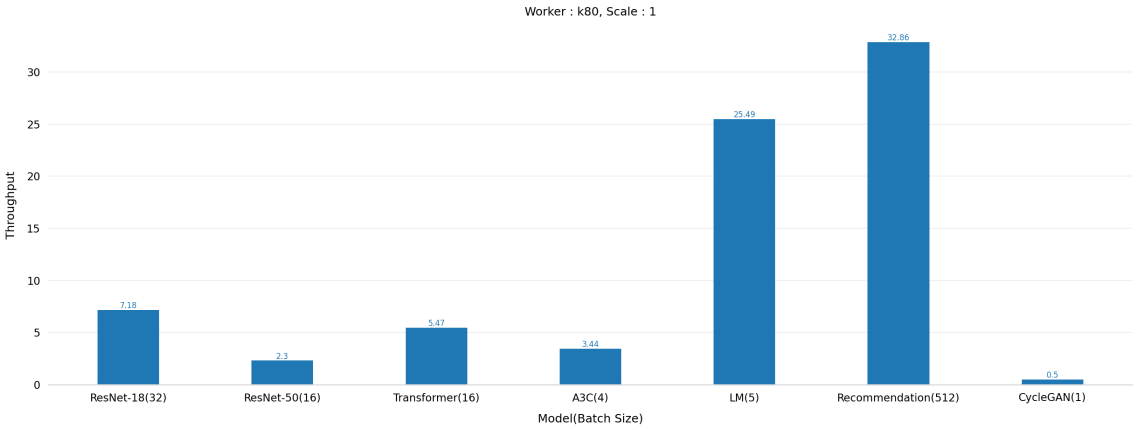

Figure 4.3: Throughput of different models for scale 1 on K80 GPU. The batch size used is the one that provides best throughput

From the graphs in this section and other graphs generated for throughput analysis we can infer the that the increase in batch-size results in a decrease in the throughput values. Theoretically speaking, the throughput should increase as the number of batch-size increases, however, this is not the case since the GPU has limited amount of memory and the Deep Learning Training jobs generate and use model weights after each iteration. The task of saving and reassign the weights is time consuming and results in a decrease in lower throughput values. Various GPUs have different architectures and the CNN and RNN models perform a lot of computation, so the throughput is also

dependent on these GPU architectures. PyTorch and TensorFlow use the GPU to their advantages and consider these architectures to improve the throughputs of each job. The model parameters such as the learning rate also have some impact on the processing speed of the various Deep Learning models.

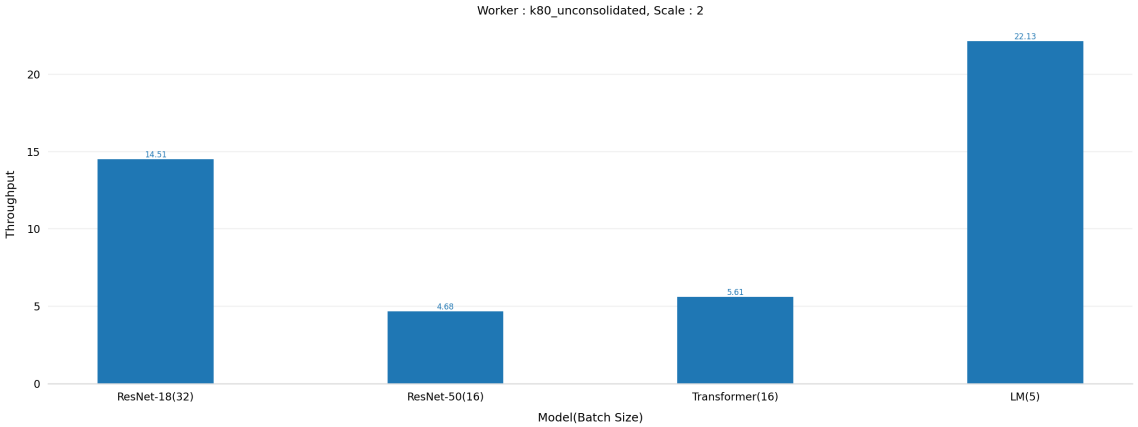

Figure 4.4: Throughput of different models for scale 2 on K80 GPU. The batch size used is the one that provides best throughput. This is the unconsolidated configuration, where the GPU are present on different servers

Another trend that we see is the increase of throughput as the scale increases. As the scale increases the number of GPUs increases, so it is quite relevant that the throughput would increase. However, we see that for some cases such as LM model for batch size 5 to 10 and also transformer, this trend is not followed and the throughput decreases as we increase the scale from 1 to 2. We can hypothesis that there are certain models which are placement dependent due to the requirements of updating and copying the large number of parameters in the form of weight files. This results in an increase in the overhead of communication and can be attributed towards the lowering of throughput. We can also find that certain Deep Learning models are time dependent and hence, would only proceed if the previous steps have all been completed. This would result in the GPU waiting for the job, thereby reducing throughput.

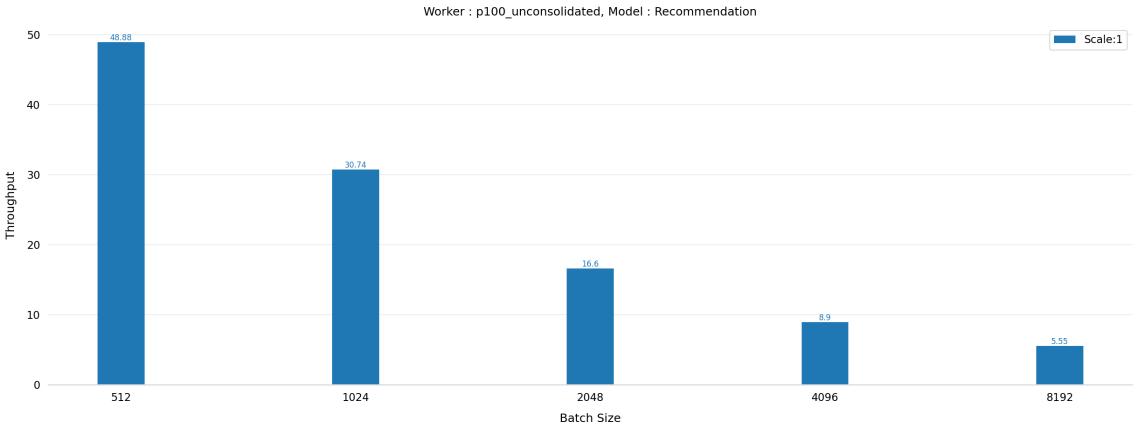

Figure 4.5: Throughput of Autoencoder on P100 GPU in case of unconsolidated configuration

We can see another trend that the throughput increases more for higher batch sizes when the number of GPUs is increased. We can explain this trend based on the logic of parallel computing. Parallel processing with multiple GPUs is an important step in scaling training of deep models. In each training iteration, typically a small subset of the dataset, called a mini-batch, is processed. When a single GPU is available, processing of the mini-batch in each training iteration is handled by this GPU. When training with multiple GPUs, the mini-batch is split across available GPUs to evenly spread the processing load. To ensure that you fully use each GPU, you must increase the mini-batch size linearly with each additional GPU. Mini-batch size has an impact not only on training speed, but also on the quality of the trained model. This effect can be seen more prominently in case of V100 GPU from scale 4 to 8, and from scale 2 to 4. Also, with higher batch sizes, the mini-batches for parallel processing can be created with ease due to possibility of dividing the dataset. If we have a lower batch size, we cannot always break the dataset to fit into a proper subset and some GPU resources would be wasted or work on sub-optimal batch sizes.

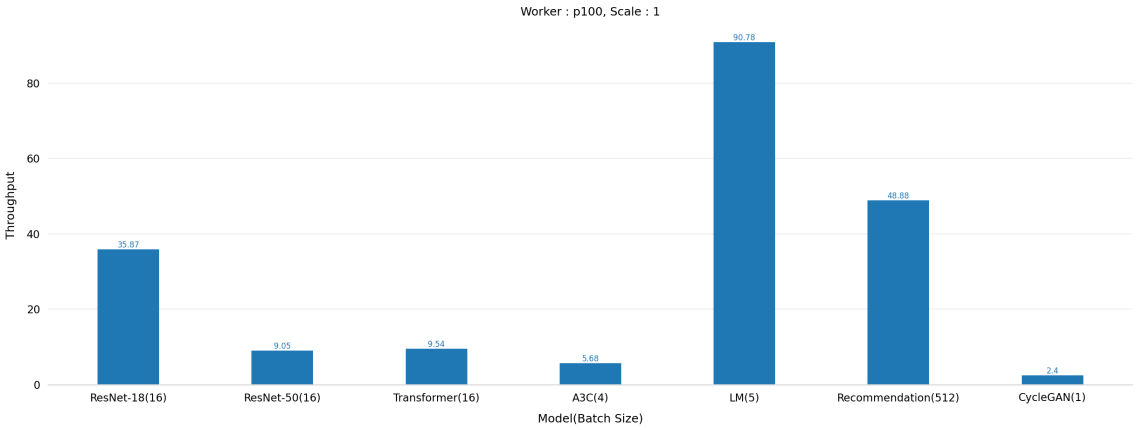

Figure 4.6: Throughput of all models for their best batch size on a P100 GPU for scale 1

We have used the resources [18], [22c], [22d], [Say21], [Muj22], [22e], [Cha20], [22f], [Rot20], [22g] to support and explain the points mentioned in the throughput analysis.

4.2 Cluster Utilization Analysis

In this section we analyze the cluster utilization of various policies implemented in the Gavel [NSK+20]. We also analyze the average completion time of different policies. Both of the analysis mentioned above are performed on the results obtained after running static and continuous sweep experiments in the gavel simulator. The bar graphs are generated for individual policies on different clusters and also all policies together, however please note this report only shows a few representative graphs. To see all the graphs, please check the Notebook [22q] files in the analysis folder at [22m].

Continuous Trace The bar graphs of continuous trace show us the comparison with the values of lambda (λ), which is used to generate the inter-arrival time between arriving jobs according to the Poison distribution process [NSK+20]. The higher the value of λ, the greater is the inter-arrival times, i.e., the jobs arrive with a long gap between them. This gives some respite to the cluster as there are often more GPU resources than the jobs. In conclusion, it can be said that higher the value

of λ, the cluster utilization will be lower. Also, λ is directly proportional to the cluster load. For the continuous trace experiments we calculate the cluster utilization and average completion time for the jobs in the window from 500 to 1000 to allow the GPU resources to be well warmed-up.

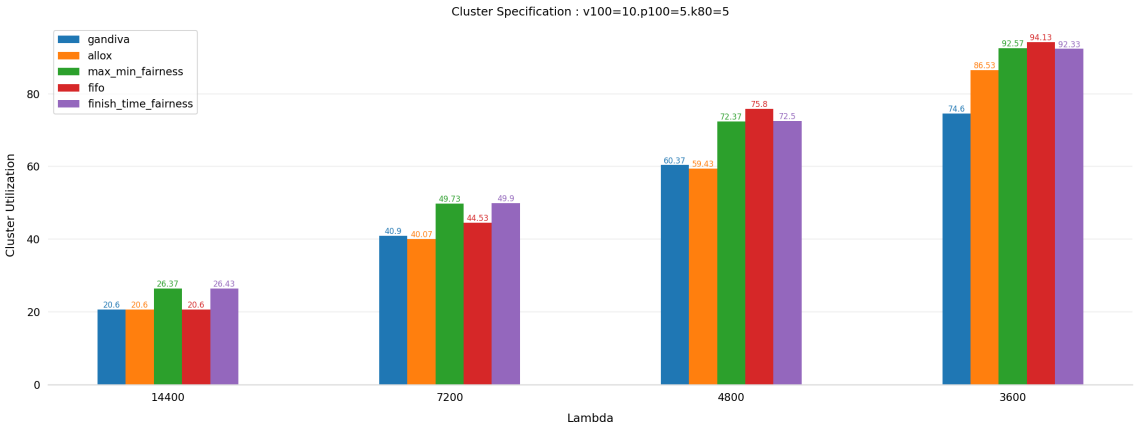

Figure 4.7: Utilization vs Lambda (Cluster Specification: v100=10, p100=5, k80=5)

Table 4.1: Cluster utilization and Average completion time for each Policy in Continuous trace.

Policy	**Cluster utilization** %	**Average completion time** hours
Gandiva	84	39
AlloX	88	26
Finish-time-fairness	95	67
Max-min-fairness	95	88
FIFO	96	130

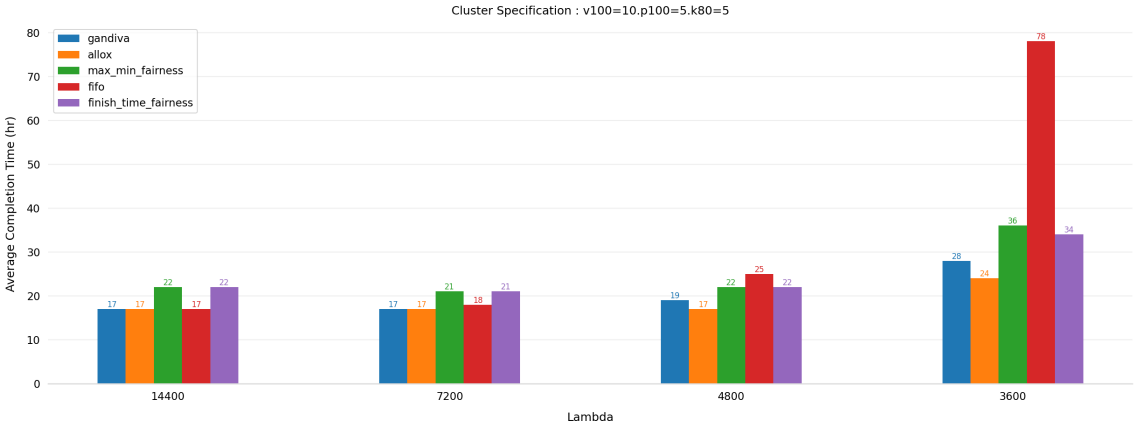

Figure 4.8: Average Completion Time vs Lambda (Cluster Specification: v100=10, p100=5, k80=5)

The figure 4.8 shows the representation of cluster utilization with lambda (λ) for the cluster with 10 V100 GPUs, 5 P100 GPUs and 5 K80 GPUs. We can clearly see that as the load on the cluster increases, there is a decrease in the cluster utilization. This is in accordance to the fact stated above that there are more resources than the jobs available, so the resources sit idle. The other cluster configurations involving different number of P100, V100 and K80 GPUs show us similar trend for the cluster utilization. From all the generated graphs, we can infer that all the policies have a higher utilization for the cluster with more K80s than V100s and P100s. One of the reasons we can argue for this is that the running time of all the jobs is higher on a K80. This can be seen from the average job completion time graphs, where the average completion times for the jobs running in this particular configuration are much higher compared to other configurations. Just like utilization, the average completion time of jobs increase with the load on the cluster. We can further infer that the jobs scheduled with the max-min-fairness, FIFO, and finish-time-fairness [MBS+19] policies take much longer than AlloX to complete the jobs. This supports our proposition that K80 GPUs are slower than P100 and V100 GPUs to complete DLT jobs. Although, the cluster utilization across different policies is the best for FIFO, the completion time is much higher as well. This indicates that FIFO can provide a good usage of our GPU cluster for the DLT jobs, but it would run for much longer duration.This is not a good solution for cloud clusters, since the users would have to pay a lot of money. Finish-time-fairness and max-min-fairness policies provide good utilization even for high loads. The table 4.1 shows the average values of utilization and average completion times over different cluster specifications for continuous trace.

Static Trace The bar graphs of static trace show us the comparison with total number of jobs at the start of the trace. These are number of DLT jobs, the particular policy must run in order to finish the experiment. We have five 5 values of total jobs from 500 to 1000 spaced equally to run for each seed. Since, all the jobs are available from start in case of static trace, we find that the utilization for nearly all the experiments is closer to 90% or higher. However, in the case of FIFO policy, the utilization falls to around 55%. However, one striking difference for the static trace is seen in the case of configurations with more K80 GPU, where the utilization is lower as compared to other policies. For static trace DLT jos, we find AlloX [LSCL20] with the least average completion time while FIFO has the higher average completion time, This is in accordance with the continuous trace. The figure 4.9 shows the cluster utilization and the figure 4.10 shows the average completion times. The table 4.2 shows the utilization and average completion times averaged for all the cluster configurations.

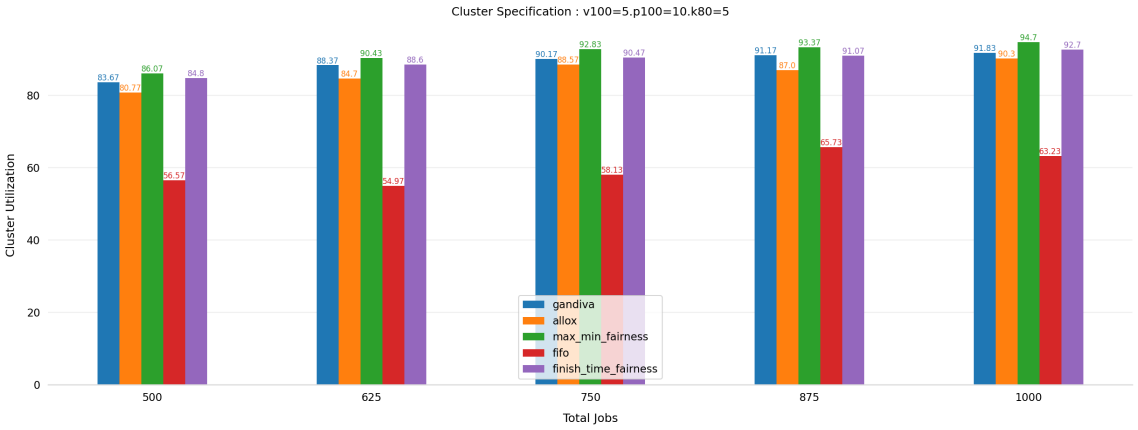

Figure 4.9: Utilization vs Total Jobs (Cluster Specification: v100=5, p100=10, k80=5)

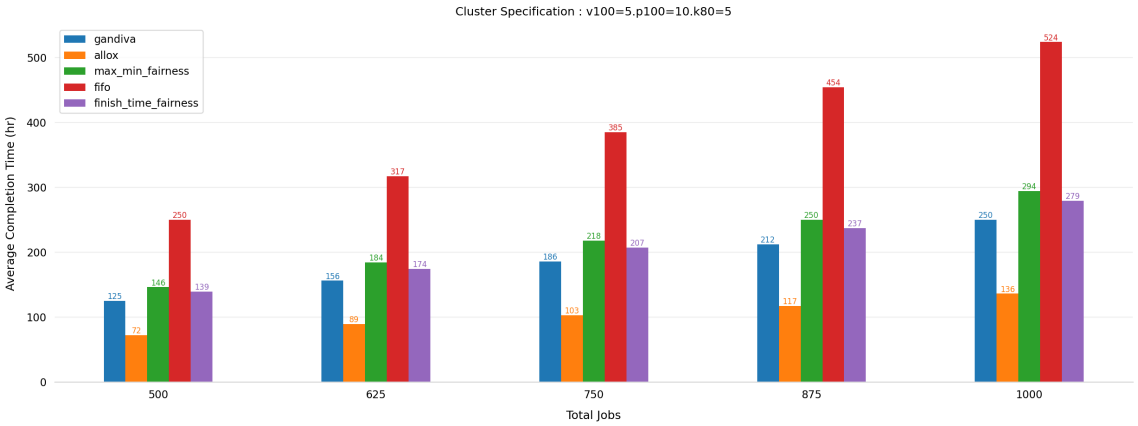

Figure 4.10: Average Completion Time vs Total Jobs (Cluster Specification: v100=5, p100=10, k80=5)

Table 4.2: Cluster utilization and Average completion time for each Policy in Static trace.

Policy	**Cluster utilization** %	**Average completion time** hours
Gandiva	92	265
AlloX	90	140
Finish-time-fairness	94	287
Max-min-fairness	95	313
FIFO	53	625

Thus, it can be inferred from the graphs showing utilization and average completion time against lambda and total jobs, that Allox [LSCL20] achieves the best job completion times, however, the utilization of cluster is not always good. Similarly for the FIFO policy, although utilization is high the completion time is too high and hence these policies do not prove to be a good fit for the cloud cluster in order to run DLT jobs. In the case of finish-time-fairness and max-min-fairness policies, the utilization along with average job completion time is moderately better. This results in cost savings for cloud clusters. We use these results to design a new metric for implementing a scheduling policy that achieves high utilization values while at the same time aims to minimize the average completion time as shown in the section 4.3. This metric is specially useful in case of cloud clusters, where cost is a important factor and as a result the job completion times need to be lower.

4.3 Shared Cost Fairness Metric

In this project, we propose a new metric shared-cost fairness, μ. This is majorly based on the finish-time fairness metric proposed by Themis [MBS+19]

$$\mu = \frac{C_{sh}}{C_{id}}$$

C_{id} is the independent cost per second and C_{sh} is the shared cost per second. The term cost here refers to the cost of using the cloud instance from three major cloud providers, namely Amazon Web Services [[22i], Google Cloud Platform [22p] and Microsoft Azure [22k]. As the trend towards cloud computing has been growing in the recent decade, we try to move all the computation to the cloud, specially the deep learning training workloads which are compute intensive and time consuming. The cloud operators charge the user based on the usage of the compute instances. In this project we aim to provide the user with a metric that would result in the improvement of using the cloud instance over time. There is a trade-off of instantaneous cost for the long time cost of using cloud instances for DLT jobs. Our new metric also takes into the account the placement of the jobs into consideration. Consider for example, if the jobs are placed in a sub-optimal way in the cluster, then the overall run-time for the jobs would increase which would increase the overall cost via the C_{sh} parameter. Given the above definition of μ, the sharing incentive for the job would be obtained if $\mu \leq 1$. To ensure this, it is necessary for the allocation mechanism to estimate the values of μ for different GPU allocations for each job.

4.4 Implementation in Gavel

4.4.1 Gavel Scheduler [NSK+20]

Given a collection of jobs, Gavel arbitrates cluster resources (in the form of accelerators of different types) among the resident jobs, while optimizing for the desired cluster objective. This is accomplished in a two-step process: first, a heterogeneity-aware policy computes the fraction of time different jobs (and combinations) should run on different accelerator types to optimize the desired objective. These policies require as input the performance behavior (in terms of throughputs) for each job on each accelerator type, which can either be provided by the user, or can be measured on the fly by Gavel's throughput estimator. Then, given the policy's output allocation, Gavel's round-based scheduling mechanism grants jobs time on the different resources, and moves jobs between workers as necessary to ensure that the true fraction of time each job spends on different resources closely resembles the optimal allocation returned by the policy. Gavel can recompute its policy either when a reset event occurs (job arrives or completes, or a worker in the cluster fails), or at periodic intervals of time.

4.4.2 Shared Cloud Cost Policy as optimization problem

Shared Cloud Cost proposes a new metric called shared-cost fairness (represented as μ), which is the ratio of the cost per second to run a job using a given allocation and the cost per second to run the job using 1/n of the cluster ($X^{isolated}$), assuming n users using the cluster. This can be expressed in terms of throughput(m, X) as follows (num_steps_m is the number of iterations remaining to train model m, c_m is the cost accumulated since the start of training for model m, and $c_m^{isolated}$ is the hypothetical cost accumulated since the start of training if model m had a dedicated fraction of the cluster to itself),

$$\mu_T(m, X) = \frac{c_m + \frac{num\_steps_m}{\frac{throughput(m,X)}{instance\_cost_m}}}{c_m^{isolated} + \frac{num\_steps_m}{\frac{throughput(m,X^{isolated})}{instance\_cost_m}}}$$

The final optimization problem is then,

$$Minimize_X \max_j \mu_T(m, X)$$

5 Algorithm

Algorithm 5.1 Shared Cloud Cost Fairness

Require: throughput
Require: timeSinceStart
Require: noRemainingSteps
Require: instanceCosts
Ensure: Base Constraint // Summation of allocation by optimization problem must not exceed the cluster specs

cumulativeIsolatedCost
stepsRemainingPrevIteration
isolatedThroughputsPrevIteration
x_allocation // allocation variable
x_isolated = getIsolatedAllocation() // isolated allocation for 1/N cluster
$isolated\_normalized\_throughput = \sum \frac{throughput \times x\_isolated}{instanceCosts}$
$expected\_cost\_fractions = []$
while job in jobs **do**
    if job not in cumulativeIsolatedCost **then**
        cumulativeIsolatedCost[job] = 0
    end if
    if job in stepsRemainingPrevIteration **then**
        $cumulativeIsolatedCost[job] = cumulativeIsolatedCost[job] + \frac{(stepsRemainingPrevIteration[job] - noRemainingSteps[job])}{isolatedThroughputsPrevIteration[job]}$
    end if
    $allocation\_normalized\_throughput = \sum \frac{throughput \times x\_allocation}{instanceCosts}$
    $expectedCostIsolated = cumulativeIsolatedCost + \frac{noRemainingSteps[job]}{isolated\_normalized\_throughput[job]}$
    $expectedCostAllocated = timeSinceStart[job] + \frac{noRemainingSteps[job]}{allocation\_normalized\_throughput}$
    $\mu = \frac{expectedCostAllocated}{expectedCostIsolated}$
    $expected\_cost\_fractions.append(\mu)$
end while
allocation = $Minimize_x \max_{job} \mu$
stepsRemainingPrevIteration = noStepsRemaining
while job in jobs **do**
    isolatedThroughputsPrevIteration[job] = isolated_normalized_throughput[job]
end while
return allocation

6 Evaluation

In this section, we seek to answer the following questions related to the our new policy, Shared Cloud Cost Fairness:

- How does our policy compare to other policies in terms of cluster utilization for both static and continuous trace?
- How does our policy compare to other policies in terms of average job completion times for both static and continuous trace?
- How does our policy compare to other policies in terms of cost involved in running DLT jobs for static trace?

6.1 Experimental Setup

6.1.1 Cluster

We run simulated cluster experiments on a 20-GPU cluster with 3 different configurations for V100s, P100s, and K80s. These configurations are as follows:

1. 5 V100s, 5 P100s, and 10 K80s
2. 5 V100s, 10 P100s, and 5 K80s
3. 10 V100s, 5 P100s, and 5 K80s

For the cost comparison of our policy with other policies we used a 15-GPU cluster with 5 V100s, 5 P100s, and 5 K80s.

6.1.2 Trace

We run simulated experiments on two types of traces:

1. **Static** All the jobs are available at the start of trace and the jobs are not added subsequently
2. **Continuous** Jobs are continuously added to the cluster. The arrival times are generated according to the Poisson arrival process initialized with an inter-arrival rate λ. Thus, showing varied cluster loads.

We use 3 fixed seeds, (5,8,9) in order to generate the same set of jobs for all the experiments. Traces are populated with a 26 job configurations as shown in the 6.1.

Model	Task	Dataset / Application	Batch size(s)
ResNet-50	Image Classification	ImageNet	16, 32, 64, 128
ResNet-18	Image Classification	CIFAR-10	16, 32, 64, 128, 256
A3C	Deep RL	Pong	4
LSTM	Language Modeling	Wikitext-2	5, 10, 20, 40, 80
Transformer	Language Translation	Multi30k (de-en)	16, 32, 64, 128, 256
CycleGAN	Image-to-Image Translation	monet2photo	1
Recoder (Autoencoder)	Recommendation	ML-20M	512, 1024, 2048, 4096, 8192

Figure 6.1: Job Configurations in Gavel

6.1.3 Metric

We use the shared cost fairness metric as defined in 4.3

6.1.4 Instance Cost

For all the experiments, we have used the spot prices of AWS instances as indicated in the prices.json file. The latest spot prices can be appended to this file. The instance p2.large indicates K80 GPU, whereas p3.2xlarge indicates v100 GPU. For P100 GPU, we multiply the cost of K80 GPU by a factor of 1.5. Since, AWS does not provide a single P100 instance. The spot prices can be updated from [22h], [22j] and [22o] for AWS, Azure and GCP, respectively.

6.1.5 Round duration

We use the round duration of 6 minutes for all experiments as suggested by Gavel [NSK+20].

6.2 End-to-End Results on Simulated Cluster

For the brevity of the report, we only show a few graph comparison. In order to view further graphs please visit the Jupyter Notebooks in the folder analysis/evaluation at [22m].

6.2.1 How does our policy compare to other policies in terms of cluster utilization for both static and continuous trace?

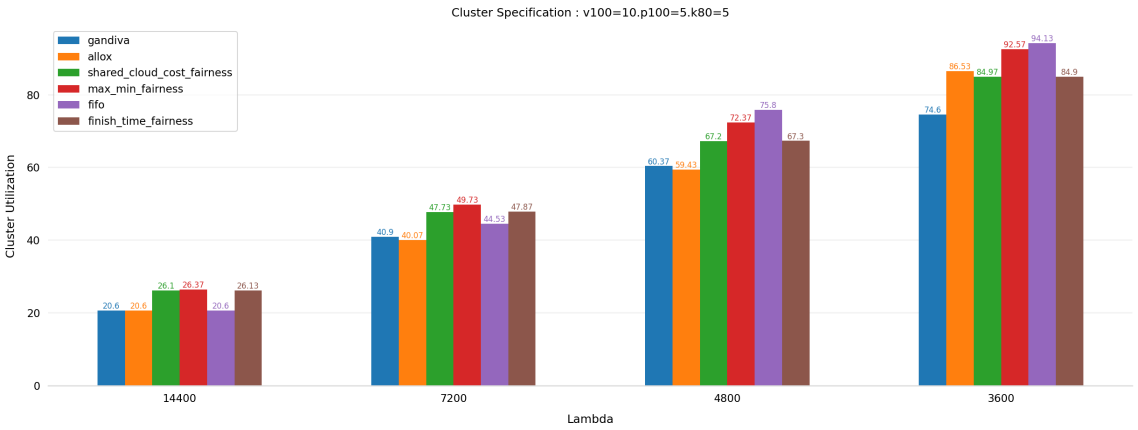

Figure 6.2: Cluster Utilization vs Lambda (Cluster Configuration:v100=10, p100=5, k80=5)

As seen in the figure 4.7 and 4.9 the utilization of cluster is nearly same for our new policy indicated by the shared cloud cost fairness. This trend can be seen across both the static and continuous trace graphs. Another inference that we can deduce from these graphs is that the utilization of our custom policy is better than the AlloX policy. We achieve utilization in the range of 90% or more in case of static trace, which was the trend for policies such as max-min-fairness and finish-time-fairness. We also see that our new policy follows the same trend for continuous trace where the utilization goes up as the load on the cluster increases and it decreases as the load decreases.

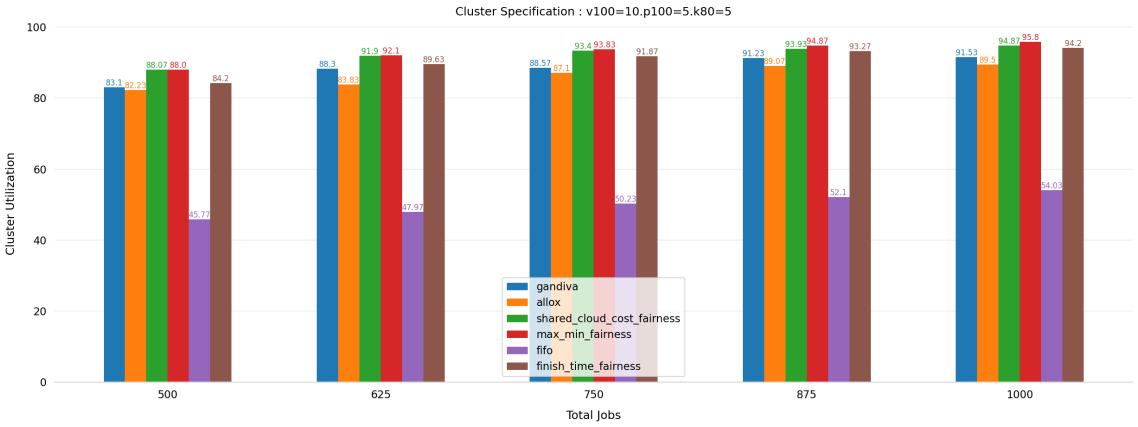

Figure 6.3: Cluster Utilization vs Total Jobs (Cluster Configuration:v100=10, p100=5, k80=5)

6.2.2 How does our policy compare to other policies in terms of average job completion times for both static and continuous trace?

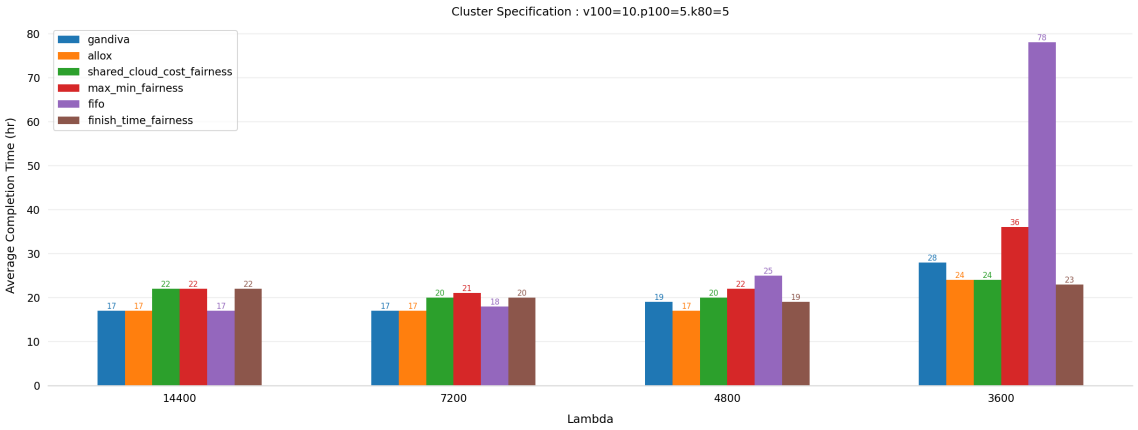

Figure 6.4: Average Completion Time vs Lambda (Cluster Configuration:v100=10, p100=5, k80=5)

With the help of figure 6.4 we can conclude that the average job completion time varies based on the load for continuous trace, with lower loads resulting in completion times of 20-25 hours whereas for higher loads this can increase to around 30-40 hours. This is still lower as compared to the other policies such gandiva, FIFO and max-min-fairness. The completion time for the cluster with higher K80 GPUs is the worst as is the case with all the other polices. For the static trace, the average completion time for our policy provides improvement from policies like FIFO and max-min-fairness.This trend follows for different cluster specifications and can be seen in the figure 6.5.

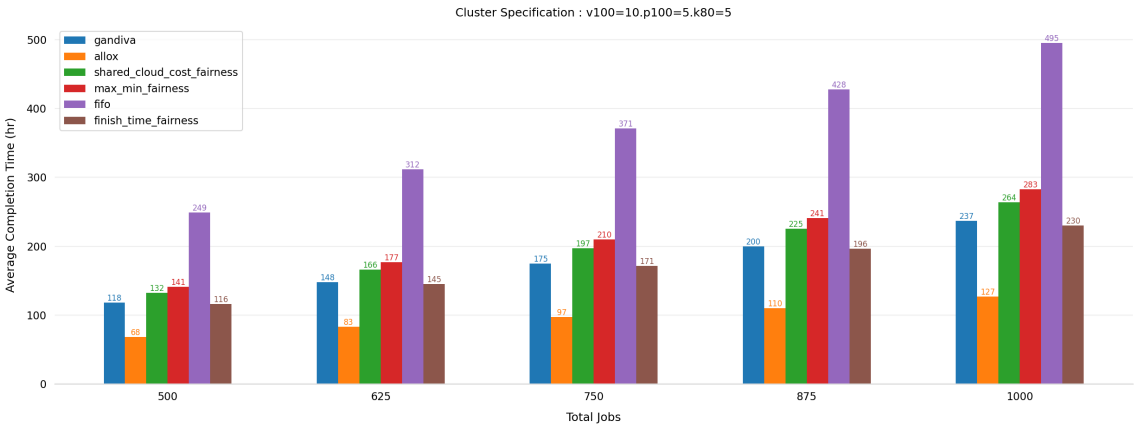

Figure 6.5: Average Completion Time vs Total Jobs (Cluster Configuration:v100=10, p100=5, k80=5)

6.2.3 How does our policy compare to other policies in terms of cost involved in running DLT jobs for static trace?

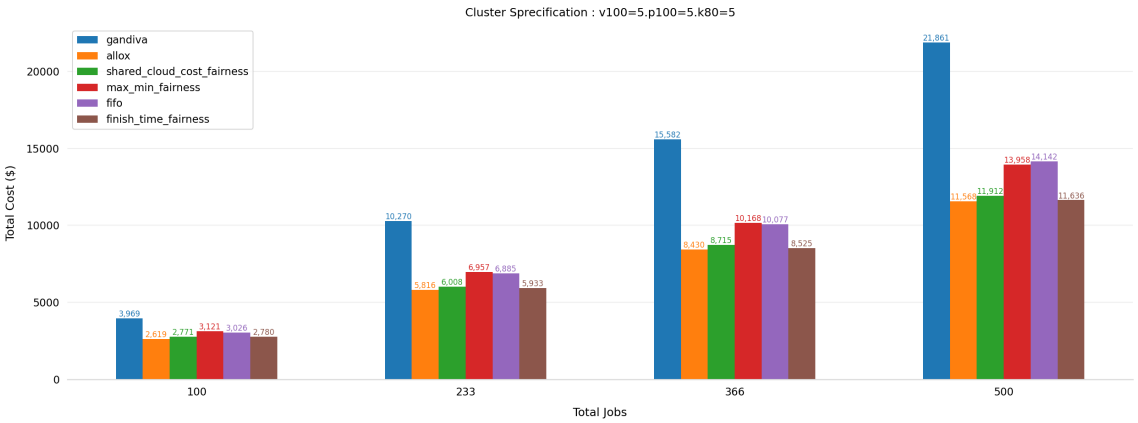

Figure 6.6: Total Cost vs Total Jobs for all policies

The average cost of running our policy for the different cluster specification can be inferred from the figure 6.7. We see that the average cost is more for the cluster configuration with more K80 GPU, since the completion time was more in this case. For 500 jobs, we achieve the cost of around 12000$ based on the cost of accelerators we specified in our prices.json file. This experiment was only done using AWS as the cloud provider with a set of fixed prices, these can be updated and we can also compare with other cloud providers. In comparison to the other policies implemented in gavel, we see that our policy achieves significant improvements in terms of cost over the policies like gavel, max-min-fairness and FIFO. We also see that our policy shows nearly the same expense as AlloX and finish-time-fairness. The comparison of our policy with other policies in terms of cost can be seen in the figure 6.6.

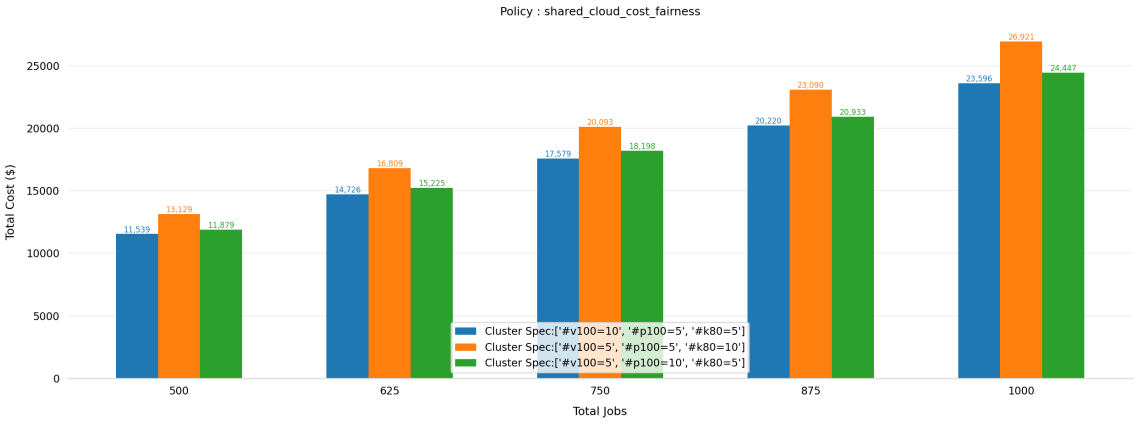

Figure 6.7: Total Cost vs Total Jobs for Shared Cloud Cost Fairness

In conclusion we find that our new policy provides an average utilization of 91% across different cluster configurations for high loads in continuous trace for 500 DLT jobs. The average completion time in the case of high loads on the continuous trace for 500 jobs is around 42 hours. For static

trace, if we consider 1000 jobs at startup then we achieve the utilization of 93% and the average completion time of 288 hours. These results are pretty much in sync with the other policies such as finish-time-fairness, max-min-fairness and gandiva. However, our policy achieves improvement in case of total costs for using the cluster. This leads us to a very significant conclusion, as the usage of cloud increases and with the advent of new DLT jobs which are ever increasing, our policy can result in saving large amounts of money while providing high utilization of the cluster and lower completion times. The average cost of running 1000 DLT jobs in static trace for different cluster specifications comes to around $25000.

7 Conclusion

In this project, firstly we analysed the throughput of running different DLT jobs on 3 different types of GPUs, namely v100, p100 and k80, to conclude that the throughput of each job depends on the resource on which the job is running. The factors that impact throughput are GPU architecture, model architecture, number of parameters in the model and also location of GPU in the cluster. Next, we analysed the cluster utilization for running a set of jobs in the cluster consisting the mentioned three types of GPUs in a particular configuration by running simulated experiments. These experiments showed that the cluster utilization is one of the important factors for usage in cloud clusters. We found that different policies with different objectives can vary their cluster utilization, but we also need to consider the job completion times. Specifically for FIFO we saw that the utilization was good, however the completion time was worst. Lastly, we proposed and implemented a new scheduling policy, Shared Cloud Cost Fairness in the gavel framework. With the simulated experiments on this new policy and its comparison with the already implemented policies in the Gavel, we found that our policy minimizes the overall cost spent on running DLT jobs on cloud instances while achieving a slight improvement in the cluster utilization.

8 Future Work

Future work based on this project could involve the following tasks:

- Running the simulated experiments on actual cluster
- Modification to include power usage in the metric
- Improve the metric based on heuristics to include operational costs such as maintenance, etc.

Bibliography

[17] *LogicMonitor, Cloud Vision 2020: The Future of the Cloud, 2019*. en-US. Dec. 2017. URL: https://www.logicmonitor.com/resource/the-future-of-the-cloud-a-cloud-influencers-survey (cit. on p. 13).

[18] en-US. June 2018. URL: https://aws.amazon.com/blogs/machine-learning/the-importance-of-hyperparameter-tuning-for-scaling-deep-learning-training-to-multiple-gpus/ (cit. on p. 26).

[22a] Python. Oct. 2022. URL: https://github.com/pytorch/examples (cit. on p. 24).

[22b] Python. Oct. 2022. URL: https://github.com/floydhub/word-language-model (cit. on p. 24).

[22c] Oct. 2022. URL: https://ai.stackexchange.com/questions/17424/effect-of-batch-size-and-number-of-gpus-on-model-accuracy (cit. on p. 26).

[22d] Oct. 2022. URL: https://towardsdatascience.com/how-to-scale-training-on-multiple-gpus-dae1041f49d2 (cit. on p. 26).

[22e] Oct. 2022. URL: https://www.reddit.com/r/learnmachinelearning/comments/j0viin/why_train_throughput_decreases_when_minibatch/ (cit. on p. 26).

[22f] Oct. 2022. URL: https://stats.stackexchange.com/questions/384198/why-increasing-the-batch-size-has-the-same-effect-as-decaying-the-learning-rate (cit. on p. 26).

[22g] en. June 2022. URL: https://blog.paperspace.com/gpu-memory-bandwidth/ (cit. on p. 26).

[22h] *AWS Pricing Website*. Oct. 2022. URL: https://aws.amazon.com/ec2/spot/pricing/?did=ap_card&trk=ap_card (cit. on p. 36).

[22i] *AWS Website*. Oct. 2022. URL: https://aws.amazon.com/ (cit. on p. 30).

[22j] *Azure Pricing Website*. en. Oct. 2022. URL: https://azure.microsoft.com/en-us/pricing/details/batch/ (cit. on p. 36).

[22k] *Azure Website*. en. Oct. 2022. URL: https://azure.microsoft.com/en-us/ (cit. on p. 30).

[22l] *Capital One*. en. Oct. 2022. URL: https://www.capitalone.com/tech/cloud/what-is-a-cluster/ (cit. on pp. 13, 15).

[22m] *Custom Code Github Repository*. Oct. 2022. URL: https://github.com/piyush-bajaj/gavel/tree/rp-submission (cit. on pp. 3, 23, 26, 36).

[22n] *Gavel Github Repository*. Oct. 2022. URL: https://github.com/stanford-futuredata/gavel (cit. on p. 13).

[22o] *GCP Pricing Website*. en. Oct. 2022. URL: https://cloud.google.com/compute/gpus-pricing (cit. on p. 36).

[22p] *GCP Website*. en. Oct. 2022. URL: https://cloud.google.com/gcp (cit. on p. 30).

[22q] *Jupyter Notebook*. en. Oct. 2022. URL: https://jupyter.org (cit. on p. 26).

[ABC+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng. "TensorFlow: a system for large-scale machine learning". In: *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*. OSDI'16. USA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1 (cit. on p. 17).

[BGO+16] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes. "Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade". In: *Queue* 14.1 (Jan. 2016), pp. 70–93. ISSN: 1542-7730. DOI: 10.1145/2898442.2898444. URL: https://doi.org/10.1145/2898442.2898444 (cit. on p. 16).

[Cha20] D. Chang. *Effect of Batch Size on Neural Net Training*. en. Aug. 2020. URL: https://medium.com/deep-learning-experiments/effect-of-batch-size-on-neural-net-training-c5ae8516e57 (cit. on p. 26).

[DK] C. Delimitrou, C. Kozyrakis. "Quasar: Resource-efficient and QoS-Aware Cluster Management". en. In: (), p. 17 (cit. on p. 17).

[GCS+19] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, C. Guo. "Tiresias: a GPU cluster manager for distributed deep learning". In: *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*. NSDI'19. USA: USENIX Association, Feb. 2019, pp. 485–500. ISBN: 978-1-931971-49-2 (cit. on pp. 13, 16, 17).

[HZRS15] K. He, X. Zhang, S. Ren, J. Sun. "Deep Residual Learning for Image Recognition". In: arXiv:1512.03385 (Dec. 2015). arXiv:1512.03385 [cs]. URL: http://arxiv.org/abs/1512.03385 (cit. on p. 24).

[KH22] W. Khallouli, J. Huang. "Cluster resource scheduling in cloud computing: literature review and research challenges". en. In: *The Journal of Supercomputing* 78.5 (Apr. 2022), pp. 6898–6943. ISSN: 1573-0484. DOI: 10.1007/s11227-021-04138-z. URL: https://doi.org/10.1007/s11227-021-04138-z (cit. on p. 15).

[kua22] kuangliu. *Train CIFAR10 with PyTorch*. Python. Oct. 2022. URL: https://github.com/kuangliu/pytorch-cifar (cit. on p. 24).

[Kuh55] H. W. Kuhn. "The Hungarian method for the assignment problem". en. In: *Naval Research Logistics Quarterly* 2.1–2 (1955), pp. 83–97. ISSN: 1931-9193. DOI: 10.1002/nav.3800020109. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109 (cit. on p. 18).

[LBH15] Y. LeCun, Y. Bengio, G. Hinton. "Deep learning". en. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539. URL: http://www.nature.com/articles/nature14539 (cit. on p. 16).

[LSCL20] T. N. Le, X. Sun, M. Chowdhury, Z. Liu. "AlloX: compute allocation in hybrid clusters". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys '20. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 1–16. ISBN: 978-1-4503-6882-7. DOI: 10.1145/3342195.3387547. URL: https://doi.org/10.1145/3342195.3387547 (cit. on pp. 13, 16, 18, 28, 29).

[MBS+19] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, S. Chawla. "Themis: Fair and Efficient GPU Cluster Scheduling". In: arXiv:1907.01484 (Oct. 2019). arXiv:1907.01484 [cs]. URL: http://arxiv.org/abs/1907.01484 (cit. on pp. 13, 16, 17, 19, 28, 29).

[Mou18] A. Moussawi. "Towards Large Scale Training Of Autoencoders For Collaborative Filtering". In: arXiv:1809.00999 (Oct. 2018). arXiv:1809.00999 [cs, stat]. DOI: 10.48550/arXiv.1809.00999. URL: http://arxiv.org/abs/1809.00999 (cit. on p. 24).

[Muj22] M. Mujtaba. *Multi GPU Model Training: Monitoring and Optimizing*. en-US. May 2022. URL: https://neptune.ai/blog/multi-gpu-model-training-monitoring-and-optimizing (cit. on p. 26).

[Nie17] J. Niemiec. *YARN - The Capacity Scheduler*. en-US. Dec. 2017. URL: https://blog.cloudera.com/yarn-capacity-scheduler/ (cit. on p. 17).

[NSK+20] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, M. Zaharia. "Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads". In: arXiv:2008.09213 (Aug. 2020). arXiv:2008.09213 [cs]. URL: http://arxiv.org/abs/2008.09213 (cit. on pp. 13, 16, 17, 26, 30, 36).

[Orl93] J. B. Orlin. "A Faster Strongly Polynomial Minimum Cost Flow Algorithm". In: *Operations Research* 41.2 (1993), pp. 338–350. ISSN: 0030-364X. URL: https://www.jstor.org/stable/171782 (cit. on p. 18).

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: arXiv:1912.01703 (Dec. 2019). arXiv:1912.01703 [cs, stat]. DOI: 10.48550/arXiv.1912.01703. URL: http://arxiv.org/abs/1912.01703 (cit. on p. 17).

[Rot20] R. Rotenberg. *How to Break GPU Memory Boundaries Even with Large Batch Sizes*. en. Jan. 2020. URL: https://towardsdatascience.com/how-to-break-gpu-memory-boundaries-even-with-large-batch-sizes-7a9c27a400ce (cit. on p. 26).

[Say21] D. Says. *Scaling Language Model Training to a Trillion Parameters Using Megatron*. en-US. Apr. 2021. URL: https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/ (cit. on p. 26).

[VMD+13] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, E. Baldeschwieler. "Apache Hadoop YARN: yet another resource negotiator". In: *Proceedings of the 4th annual Symposium on Cloud Computing*. SOCC '13. New York, NY, USA: Association for Computing Machinery, Oct. 2013, pp. 1–16. ISBN: 978-1-4503-2428-1. DOI: 10.1145/2523616.2523633. URL: https://doi.org/10.1145/2523616.2523633 (cit. on p. 16).

[XBR+18] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, L. Zhou. "Gandiva: Introspective Cluster Scheduling for Deep Learning". en. In: 2018, pp. 595–610. ISBN: 978-1-939133-08-3. URL: https://www.usenix.org/conference/osdi18/presentation/xiao (cit. on pp. 13, 16).

[ZBS+10] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica. "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling". In: *Proceedings of the 5th European conference on Computer systems*. EuroSys '10. New York, NY, USA: Association for Computing Machinery, Apr. 2010, pp. 265–278. ISBN: 978-1-60558-577-2. DOI: 10.1145/1755913.1755940. URL: https://doi.org/10.1145/1755913.1755940 (cit. on p. 17).

[ZSOF17] H. Zhang, L. Stafman, A. Or, M. J. Freedman. "SLAQ: quality-driven scheduling for distributed machine learning". en. In: *Proceedings of the 2017 Symposium on Cloud Computing*. Santa Clara California: ACM, Sept. 2017, pp. 390–404. ISBN: 978-1-4503-5028-0. DOI: 10.1145/3127479.3127490. URL: https://dl.acm.org/doi/10.1145/3127479.3127490 (cit. on p. 13).

All the links were accessed on 21-10-2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature