Advanced Scheduling Techniques for Distributed Deep Learning Training in Kubernetes Clusters

학번: 2016270337, 이름: 오종민
지도교수: 유혁 (서명)

2023.6.13

Abstract

With the advancement of deep learning models, the size and complexity of models as well as the training data have significantly increased, resulting in longer training times. To address this issue, distributed deep learning techniques are being employed, where multiple workers simultaneously train the model in a parallel manner. In cloud environments, there are four main techniques used for scheduling distributed deep learning jobs: load balancing, bin packing, gang, and co-location. In this paper, we provide a comparative analysis of these four techniques based on three criteria: 1) whether all workers are scheduled together, 2) consideration of network bottlenecks, and 3) occurrence of job waiting times. By analyzing these aspects, we aim to evaluate the effectiveness and limitations of each technique and propose future research directions to overcome their shortcomings.

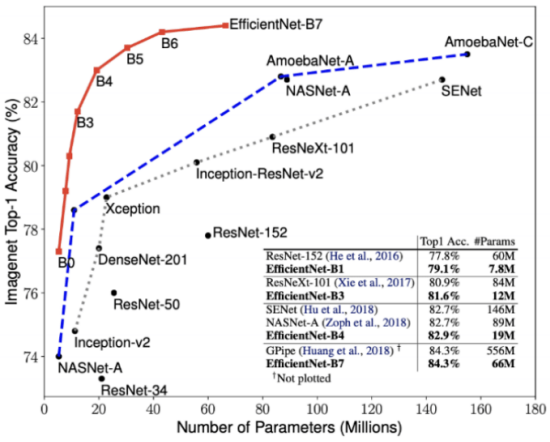

Figure 1: Incresing Model Parameter, EfficientNet[1]

1 Introduction

Deep learning models have become increasingly popular in various fields, such as computer vision, natural language processing, and speech recognition, due to their ability to learn complex patterns and representations from large-scale data. Especially, several prominent deep learning models have emerged in the fields of computer vision and natural language processing (NLP).

In computer vision, VGGNet and ResNet have gained significant attention. VGGNet is characterized by its deep network structure and the use of smaller filter sizes. It typically consists of 16 to 19 layers and has approximately 138 million parameters. VGGNet is often trained on large-scale image datasets such as ImageNet. ResNet, on the other hand, introduced residual learning to address the issue of vanishing gradients. It offers various depths of network structures, with ResNet-50 being a popular choice. ResNet-50 has around 25 million parameters and is typically trained on datasets like ImageNet.

Moving on to NLP, Transformer and BERT are among the most renowned models. Transformer utilizes attention mechanisms to understand sentence semantics and has been successfully applied to vari-

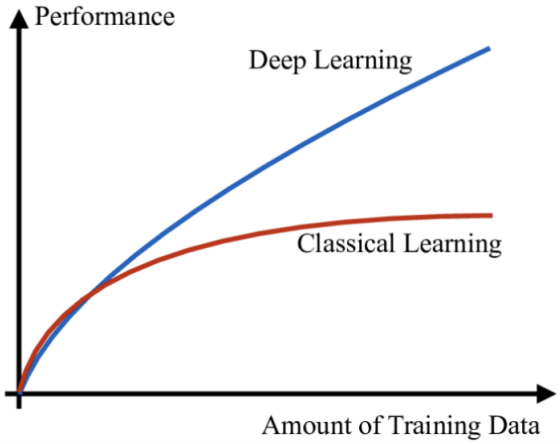

Figure 2: Incresing Train data[2]

ous NLP tasks. The complexity of Transformer models can vary, with different sizes available. For example, BERT-Base has around 110 million parameters. These models are typically pretrained on large-scale text corpora such as Wikipedia and BookCorpus. BERT, based on bidirectional Transformers, is a pretrained language model that captures contextual word embeddings. The BERT-Base model consists of approximately 110 million parameters. Pretraining is performed on extensive text datasets, followed by fine-tuning on specific tasks using smaller datasets.

However, as the size and complexity of these models grow, training them on a single machine becomes infeasible due to resource constraints and lengthy training times. To overcome these limitations, distributed training has emerged as a promising solution that leverages multiple machines to collaboratively train deep learning models.

Distributed deep learning refers to the approach of training models on multiple computer nodes. Several training methods have been developed for distributed deep learning, and one important approach is the Parameter Server method.

The Parameter Server method involves a central Parameter Server node(Parameter Server) and multiple Worker nodes(Worker) in a cluster. The Worker divide the data and perform parallel training, sending the computed gradients to the Parameter Server. The Parameter Server manages and updates the model parameters. This approach operates by exchanging gradients through network communication, effectively utilizing distributed computing resources.

The key advantage of the Parameter Server method is the separation of data and model parameters, reducing network overhead. However, it may introduce communication overhead concentrated on the Parameter Server and can be affected by failures of the Parameter Server node, impacting the overall system performance. Therefore, efficient network communication and a reliable Parameter Server are crucial for this approach.

Distributed training methods like the Parameter Server approach are commonly utilized in cloud environments. To ensure efficiency in the process of creating and deploying worker nodes, orchestration platforms are employed. Among the widely used platforms, Kubernetes(k8s) stands out as a popular choice, particularly for container-based deployments.

With k8s, workers are generated and deployed on physical servers based on the resource requirements and environment of each distributed deep learning task. The decision of worker placement on servers is determined by scheduling techniques, or schedulers, which include the k8s native scheduler, the Volcano scheduler, or custom schedulers within the k8s ecosystem.

This paper aims to compare and analyze the scheduling techniques primarily used in distributed deep learning in cloud environments. Through this analysis, the paper aims to identify the limitations and drawbacks of existing scheduling techniques and explore potential areas for future research. By examining the performance, efficiency, and scalability of different scheduling methods, the paper seeks to provide insights into the challenges and opportunities in improving the deployment and management of distributed deep learning tasks in cloud environments.

2 Background

This section briefly explain the structure of the k8s cluster and then, introduce the scheduling techniques

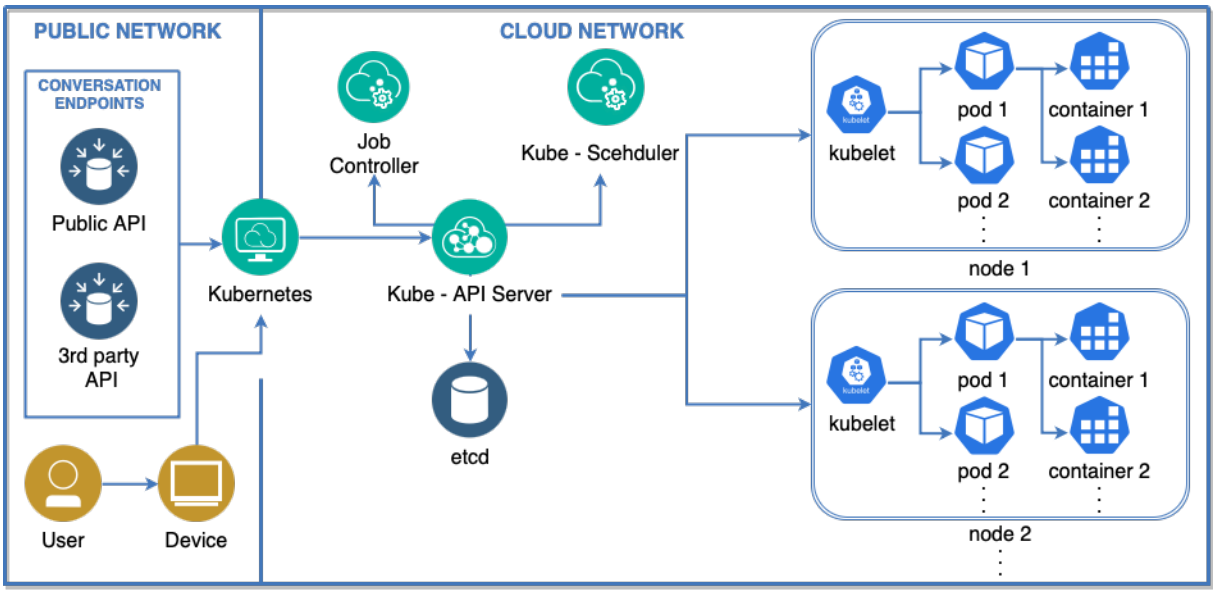

Figure 3: k8s Cluster Arichitecture

used in distributed DLT in k8s clusters, including the k8s native scheduler, the Volcano scheduler, and custom schedulers.

2.1 K8s Cluster Architecture

Each component in Kubernetes has a specific role and function in the management and orchestration of containerized applications within a cluster.

2.1.1 User

The user interacts with k8s through various command-line tools or graphical interfaces. Users can issue commands to create, modify, or delete k8s resources, such as Pods, Services, and Deployments. The user's commands are sent to the API Server for processing.

2.1.2 kube-API Server

The API Server in **Figure 3** is the central control plane component in k8s. It exposes the k8s API, allowing users to interact with the cluster. The API Server authenticates and authorizes incoming requests from users. It validates and processes the API requests, maintaining the desired state of the cluster. The state of the cluster is stored in etcd.

2.1.3 kube-scheduler

The kube-scheduler is responsible for determining the optimal placement of Pods on available nodes in the cluster. It takes into account factors such as resource requirements, node affinity, and anti-affinity, as well as user-defined constraints. The scheduler selects a suitable node for each Pod based on these factors and updates the Pod's assignment information in the API Server. It follows a multi-step process involving filtering, scoring, and binding to determine the optimal placement of Pods. Here's a detailed explanation of each step:

1. Filtering:

- The kube-scheduler starts by filtering out nodes that are unsuitable for running the Pod. It does this by considering factors such as resource avail-

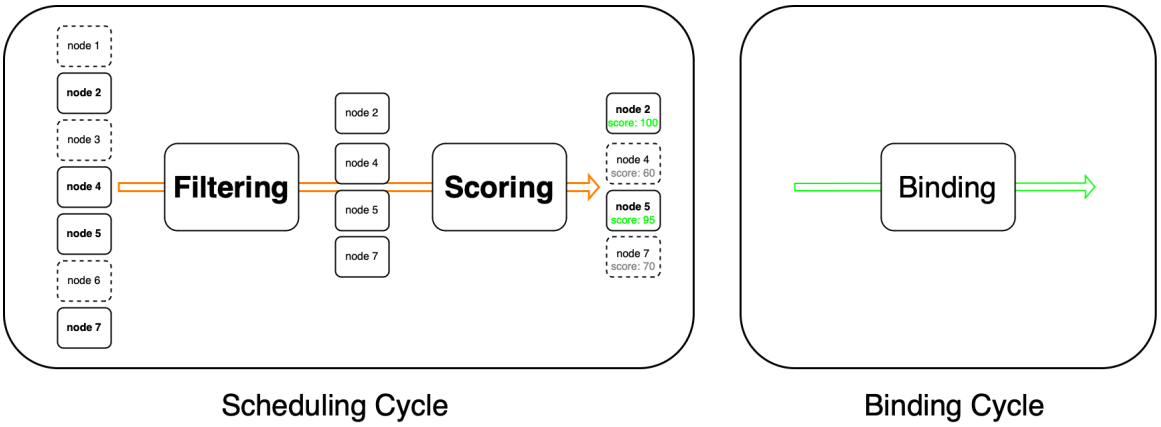

Figure 4: k8s Scheduler Workflow

ability, node conditions, affinity rules, taints, and tolerations.

- The filtering process eliminates nodes that don't meet the Pod's requirements, ensuring that only eligible nodes are considered for scheduling.

2. Scoring:

- After filtering, the kube-scheduler assigns a score to each remaining node. The score reflects the desirability of the node for hosting the Pod.
- The scoring algorithm takes into account various factors, including resource utilization, inter-pod affinity or anti-affinity rules, locality constraints, node capacity, and other user-defined preferences.
- Nodes with higher scores are considered more suitable for running the Pod.

3. Binding:

- Once the scoring is complete, the kube-scheduler selects the node with the highest score as the preferred node for scheduling the Pod.
- The kube-scheduler then creates a binding between the Pod and the selected node, ensuring that the Pod is assigned to the chosen node for execution.
- If the binding process fails (e.g., due to node unavailability or resource constraints), the kube-scheduler retries the process with the next highest-scoring node until a successful binding is achieved.

By performing filtering, scoring, and binding, the kube-scheduler determines the best node for running a Pod based on resource requirements, constraints, and user preferences. This ensures efficient resource allocation, improved performance, and adherence to application-specific requirements in the Kubernetes cluster.

2.1.4 kube-controller-manager

The kube-controller-manager consists of various controllers that manage different aspects of the cluster. In this paper, we focus on the Job Controller in **Figure 3**, which is responsible for managing the execution and lifecycle of distributed DLT Jobs in the cluster. The Job Controller plays a crucial role in managing batch workloads and running one-time tasks in k8s. It provides the necessary functionality to create, monitor, and track the progress of Jobs, ensuring reliable execution and automatic cleanup of resources.

Here's how the Job Controller operates:

- **Job Creation:** When a user creates a Job object, specifying the desired number of completions and the Pod template for the Job Pods, the Job Controller receives this request.
- **Pod Creation:** The Job Controller creates one or more Pod instances, known as Job Pods, based on the specified Pod template. Each Job Pod represents an individual task or unit of work to be executed.
- **Pod Scheduling:** The Job Controller ensures that the Job Pods are scheduled to run on available worker nodes in the cluster. It considers factors such as resource requirements, affinity rules, and node availability during the scheduling process.
- **Pod Monitoring:** As the Job Pods are running, the Job Controller continuously monitors their status and progress. It checks if the Job Pods have completed successfully or if any failures or errors have occurred.
- **Completion Tracking:** The Job Controller tracks the number of successful completions achieved by the Job Pods. It compares this count with the desired number of completions specified in the Job object.
- **Job Status Updates:** The Job Controller updates the status of the Job object, reflecting the current state of the Job Pods and the number of successful completions. It provides information on the progress of the Job execution.
- **Job Termination:** Once the desired number of completions is reached or if the Job fails to meet the completion criteria within the specified tolerances, the Job Controller terminates the running Job Pods.
- **Cleanup:** After the Job is completed or terminated, the Job Controller cleans up any remaining Job Pods and associated resources. This ensures efficient resource utilization within the cluster.

Each controller including Job controller continuously watches the cluster's state through the API Server and takes actions to maintain the desired state. Controllers reconcile the actual state with the desired state, creating, updating, or deleting resources as needed.

2.1.5 kubelet

The kubelet runs on each node in the cluster and manages the Pods and containers on that node. It communicates with the API Server to receive instructions and report the status of the node. The kubelet is responsible for pulling container images, starting and stopping containers, and monitoring their health. It works in conjunction with the container runtime (e.g., Docker) to manage the containers within the Pods.

2.1.6 etcd(DB)

etcd is a distributed key-value store that serves as the cluster's source of truth. It stores the configuration and state information of the cluster. The API Server reads from and writes to etcd to maintain the desired state of the cluster. etcd ensures consistency and fault tolerance by replicating the data across multiple nodes in the cluster.

2.2 K8s Scheduling Techniques

K8s Native Scheduler

2.2.1 Load Balancing Technique

The load balancing technique operates during the scoring phase of the native scheduler in k8s. It selects a specific server among those that satisfy the resource requirements of the worker during the filtering process. The load balancing technique aims to place the worker on the server with the highest available resources. This ensures an even distribution of workers across servers, maintaining similar levels of resource utilization on each server.

2.2.2 Bin Pack Technique

The bin pack technique also operates during the scoring phase of the native k8s scheduler. In contrast to

the load balancing technique, it places the worker on the server with the lowest available resources among those that satisfy the worker's resource requirements. This reduces resource fragmentation on specific servers and maximizes resource utilization. However, this approach may result in relatively lower resource utilization on other servers, leading to an imbalance in resource utilization within the cloud environment.

Volcano Scheduler

2.2.3 Gang Technique

The Volcano scheduler is primarily used to support the gang scheduling technique, which is not natively supported by the default k8s scheduler. The general workflow of the Volcano scheduler is similar to the default scheduler. However, it incorporates both the gang and bin pack techniques to efficiently schedule distributed deep learning jobs.

In the context of Volcano, the Gang technique follows an "All or Nothing" approach. It considers all the workers required for a particular learning task at once. To apply this technique, the available resources on the servers in the cloud environment must be sufficient to accommodate the combined resource requirements of all the workers. If sufficient resources are available, the bin pack technique is used to schedule all the workers on the server with the lowest available resources. In cases where immediate scheduling is not possible, the job is placed in the waiting queue and waits for a suitable time to be scheduled.

3 Problem

Distributed training, where the computation is spread across multiple machines, can significantly reduce the training time. However, distributed training also introduces new challenges related to scheduling and communication between machines. The existing scheduling methods may not be optimal for deep learning workloads and can lead to inefficient resource utilization, which can result in longer training times and higher resource costs. Some of the major challenges in distributed deep learning training include:

- **Stragglers**: Due to the heterogeneity of the machines and the unpredictable nature of deep learning workloads, some tasks may take longer to complete than others. These tasks, called stragglers, can significantly slow down the overall training process.
- **Communication overhead**: Communication between machines can become a major bottleneck in distributed training, especially when dealing with large models and datasets. Minimizing communication overhead is crucial for improving the efficiency and scalability of distributed training.
- **Resource allocation and utilization**: Optimally allocating and utilizing resources, such as GPUs and CPUs, is essential for reducing training time and resource costs. However, existing scheduling methods may not be well-suited for the dynamic and unpredictable nature of deep learning workloads, leading to inefficient resource usage.
- **Fault tolerance**: Distributed training involves multiple machines, which increases the likelihood of machine failures. Developing fault-tolerant scheduling methods is important for ensuring the robustness and reliability of distributed training.

This paper mainly focus on 'Communication overhead' and 'Resource allocation and utilization' problem. Plus, by comparing and analyzing the scheduling techniques used in distributed deep learning training, we aim to identify the limitations and drawbacks of existing scheduling techniques and explore potential areas for future research.

4 Approach

Custom Scheduler (Co-location Technique)

The custom scheduler[3, 4] allows users to implement their own scheduling techniques that are not supported by the K8s default scheduler. In this paper, we focus on the co-location technique, which controls the network bottleneck among the workers in distributed

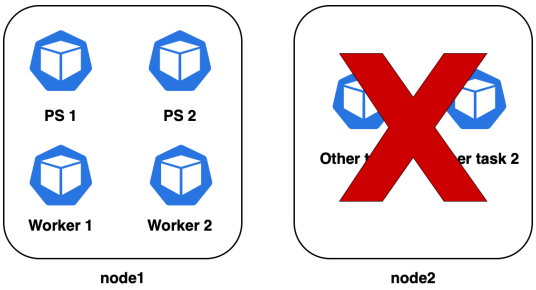

Figure 5: Co-location Technique Design

deep learning jobs by considering the communication traffic between servers.

Co-location Technique Design

Figure 5 shows the design of the co-location technique. The co-location technique involves deploying all the workers of a single training task on a single server. In distributed deep learning jobs, frequent network communication occurs for the transmission of calculated parameters and updated parameters between individual workers. When workers are divided and deployed across multiple servers, the cost of communication increases, leading to a higher possibility of network bottlenecks.

5 Evaluation

Experimental Setup

We conducted experiments to evaluate the performance of the co-location technique in comparison with the load balancing, bin pack, and gang techniques. Using 10 model with various training data(CIFAR10, ImageNet), we measured the waiting time, scheduling time, job completion time(jct), which consist of waiting time and training time, and network traffic of each technique. The experiments were conducted on a k8s cluster with 5 servers, 2 of them equipped with 4 NVIDIA Tesla V100 GPUs, 2 of another equipped with 2 NVIDIA Tesla V100 GPUs. The remaining server, which is master server, running k8s cluster. The servers are connected via a 10Gbps network.

Experimental Results

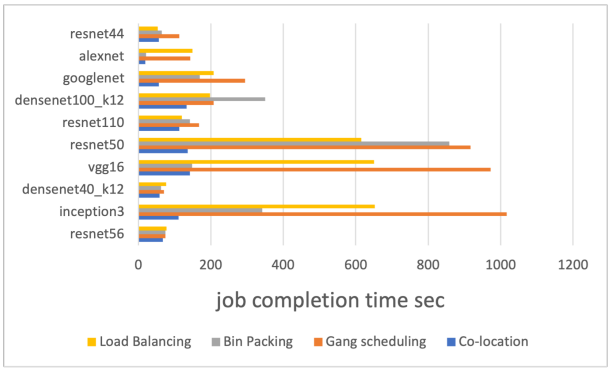

Figure 6: Job Completion Time

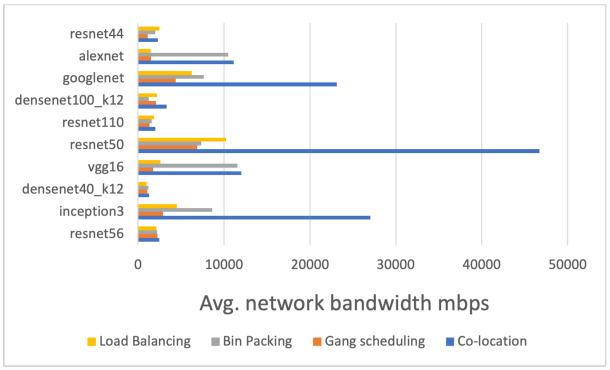

Figure 7: Avg. Network Bandwidth

Figure 7 shows the average network bandwidth of each techniques. The co-location technique has the highest network traffic among the four techniques. Since other techniques distribute workers across multiple servers, it causes network bottlenecks and reduces the performance of the distributed deep learning job. However, the co-location technique minimizes network traffic by placing all workers on a single server, reducing the occurrence of network bottlenecks. By these reasons, the co-location technique has

the lowest JCT among the four techniques. **Figure 6** shows the JCT of each techniques. we can see that the co-location technique has the lowest JCT among the four techniques.

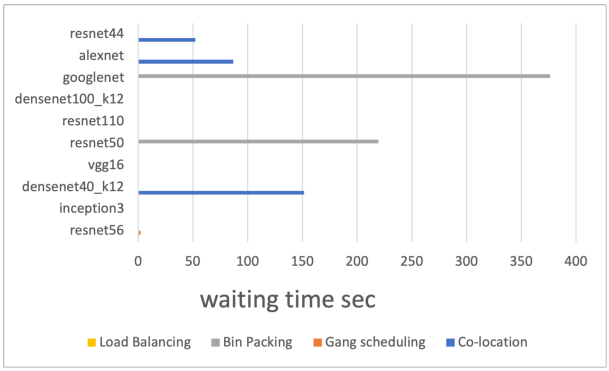

Figure 8: waiting time

However, the co-location technique may cause job waiting times if there is no server available that meets the resource requirements for all workers of the deep learning job. **Figure 8** shows the waiting time of each techniques. The co-location technique has the highest waiting time in some model among the four techniques. This is because the co-location technique requires a server with sufficient resources to accommodate all workers of a deep learning job. If such a server is not available, the job is placed in the waiting queue and waits for a suitable time to be scheduled. It is also same as Bin pack technique.

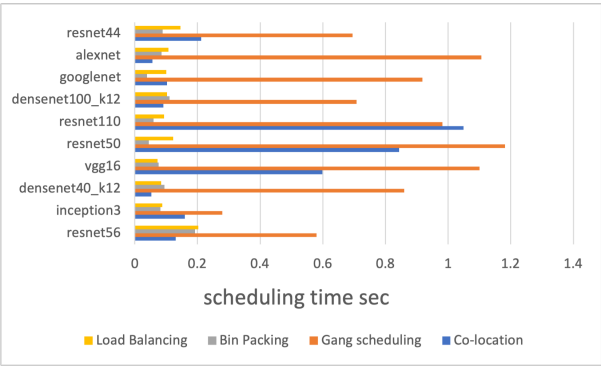

Figure 9: scheduling time

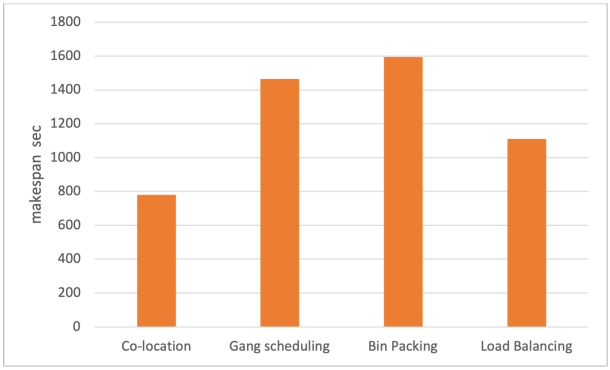

Figure 10: makespan

Figure 11: Avg. GPU Utilization

Figure 9 shows the scheduling time of each techniques. The co-location technique has the low scheduling time on average among the four techniques. But it has high scheduling time in some model. This is because the co-location technique requires a server with sufficient resources to accommodate all workers of a deep learning job.

makespan means the time from the start of the first job to the end of the last job. **Figure 10** shows the makespan of each techniques. The co-location technique has the lowest makespan among the four techniques. This is because the co-location technique minimizes network traffic by placing all workers on a single server, reducing the occurrence of network bottlenecks. **Figure 11** shows the average GPU utilization of each techniques. Since the co-location technique

places all workers of a single job on a single server, maximizing GPU utilization.

Comparison and Analysis of Distributed Deep Learning Job Scheduling Techniques

In this paper, we analyze four distributed deep learning job scheduling techniques: load balancing, bin pack, gang, and co-location. The load balancing technique selects servers with the highest available resources to evenly distribute workers and maintain balanced resource utilization. Conversely, the bin pack technique places workers on servers with the lowest available resources to minimize resource fragmentation and maximize utilization. The gang technique considers the simultaneous placement of all workers required for a single distributed deep learning job. Co-location places all workers of a single job on a single server to avoid network bottlenecks.

To compare these techniques, we analyze three criteria: 1) whether all workers are scheduled together, 2) consideration of network bottlenecks, and 3) occurrence of job waiting time.

First, the gang and co-location techniques consider and schedule all workers together. In contrast, the load balancing and bin pack techniques sequentially schedule workers, potentially leading to situations where some workers cannot be placed if there are no available resources left in the cloud environment. Workers that are already scheduled wait for the remaining workers to be placed, resulting in resource wastage.

Second, the co-location technique takes network bottlenecks into consideration, unlike other techniques. Load balancing distributes workers across multiple servers, increasing the likelihood of network bottlenecks and causing significant performance degradation. The bin pack and gang techniques also frequently result in network bottlenecks due to the placement of workers on multiple servers.

Lastly, the load balancing, bin pack, and gang techniques immediately schedule deep learning jobs if the available resources in the cloud environment satisfy the job's resource requirements. However, the co-location technique may cause job waiting times if there is no server available that meets the resource requirements for all workers of the deep learning job.

6 Conclusion

Conclusion and Futurse Research Directions

As the size of deep learning models and data increases, distributed training in cloud environments becomes essential. While various scheduling techniques exist, none of them simultaneously consider and satisfy criteria such as the utilization of GPU resources, waiting time, and network bottlenecks. For example, load balancing, bin pack, and gang techniques lead to reduced training speed due to network bottlenecks, while the co-location technique may result in job waiting times even if the overall available resources in the cloud meet the resource requirements of the deep learning job.

In the future, our research team aims to propose and develop a scheduling technique that comprehensively considers all three aspects, based on the analysis and previous research on predicting communication volumes in distributed deep learning jobs[5, 6]. We anticipate that this approach will improve resource utilization, reduce waiting time, and minimize network bottlenecks without compromising training speed.

7 Related Work

[1] MingxingTan, et al.,"EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," Proceedings of the 36th International Conference on Machine Learning,PMLR 97:6105-6114,2019.

[2] Al-Shabibi, Ali, et al. AlessioZappone, et al., "Wireless Networks Design in the Era of Deep Learning: Model-Based, AI-Based, or Both," IEEE Transactions on CommunicationsPP(99):1-1, DOI:10.1109/TCOMM.2019.2924010.

[3] M. Amaral, J. Polo, D. Carrera, S. Seelam and M. Steinder, "Topology-Aware GPU Scheduling for Learning Workloads in Cloud Environments", Proc. of SC, 2017.

[4] W. Xiao et al., "Gandiva: Introspective cluster scheduling for deep learning", 13th USENIX Sympo-

sium on Operating Systems Design and Implementation (OSDI 18), pp. 595- 610, 2018.

[5] G. Yang, C. Shin, J. Lee, Y. Yoo and C. Yoo, "Prediction of the resource consumption of distributed deep learning systems", Proc. ACM Meas. Anal. Comput. Syst, vol. 6, no. 2, 2022.

[6] Yeonho Yoo, et al., "Control Channel Isolation in SDN Virtualization: A Machine Learning Approach," in 23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid23).