Computers, Materials & Continua
DOI: 10.32604/cmc.2022.023318
Article

Tech Science Press

Multi-Agent Deep Reinforcement Learning-Based Resource Allocation in HPC/AI Converged Cluster

Jargalsaikhan Narantuya[1,*], Jun-Sik Shin[2], Sun Park[2] and JongWon Kim[2]

[1]Department of Cloud, Kakao Enterprise Corp, Seongnam, 13494, Korea
[2]AI Graduate School, Gwangju Institute of Science and Technology (GIST), Gwangju, 61005, Korea
*Corresponding Author: Jargalsaikhan Narantuya. Email: jarven.17@kakaoenterprise.com
Received: 03 September 2021; Accepted: 15 October 2021

Abstract: As the complexity of deep learning (DL) networks and training data grows enormously, methods that scale with computation are becoming the future of artificial intelligence (AI) development. In this regard, the interplay between machine learning (ML) and high-performance computing (HPC) is an innovative paradigm to speed up the efficiency of AI research and development. However, building and operating an HPC/AI converged system require broad knowledge to leverage the latest computing, networking, and storage technologies. Moreover, an HPC-based AI computing environment needs an appropriate resource allocation and monitoring strategy to efficiently utilize the system resources. In this regard, we introduce a technique for building and operating a high-performance AI-computing environment with the latest technologies. Specifically, an HPC/AI converged system is configured inside Gwangju Institute of Science and Technology (GIST), called GIST AI-X computing cluster, which is built by leveraging the latest Nvidia DGX servers, high-performance storage and networking devices, and various open source tools. Therefore, it can be a good reference for building a small or middle-sized HPC/AI converged system for research and educational institutes. In addition, we propose a resource allocation method for DL jobs to efficiently utilize the computing resources with multi-agent deep reinforcement learning (mDRL). Through extensive simulations and experiments, we validate that the proposed mDRL algorithm can help the HPC/AI converged cluster to achieve both system utilization and power consumption improvement. By deploying the proposed resource allocation method to the system, total job completion time is reduced by around 20% and inefficient power consumption is reduced by around 40%.

Keywords: Deep learning; HPC/AI converged cluster; reinforcement learning

1 Introduction

We currently live in the era of big data analysis and artificial intelligence (AI). Since AI and big data analysis are capable of processing enormous amounts of data, the demand for large-scale

computing for AI research is continuously increasing. Thus, researchers and developers are paying more attention to building a high-performance computing (HPC)-based AI-computing environment to accelerate the performance of deep learning (DL) workloads.

However, building and operating an HPC-enabled AI-computing environment not only needs multiple GPU servers, but it also requires broad knowledge to leverage the latest computing, networking, and storage technologies. For instance, understanding network interconnects (e.g., Infiniband, RoCE, and Ethernet), network channels to communicate through them (gRPC, MPI), distributed storage solutions (NFS, Ceph) are essential for scaling the AI-computing environment. Besides this, deploying the right software methods, such as Slurm and Kubernetes, for resource management and job scheduling is one of the key factors needed to improve the system performance.

In [1], NVIDIA introduced an optimized system, called the DGX SuperPod. It is configured and designed for multi-node DL and HPC. DGX SuperPod consists of 140 DGX A100 servers with 1120 NVIDIA A100 GPUs, and the DGX A100 s are connected to each other through 140 Infiniband 200G switches. The Massachusetts Institute of Technology (MIT) has another scalable AI-computing environment, called Satori. It is developed in collaboration with IBM. Satori consists of 64 IBM power nodes, nine nodes with 256 NVIDIA V100 GPUs, and 2PB storage. It has a 64TB system memory and 8TB GPU memory. GPUs inside a node are connected by an NVLink network that supports nearly 200G bi-directional transfer between GPUs [2]. However, building and monitoring such a degree of scalability is only possible for organizations with sufficient funds to deploy infrastructure at a large scale.

Additionally, the problem of batch job scheduling is a long-standing topic in supercomputing. Early studies examine the decision regarding the priority of the batch jobs. They developed various scheduling policies, such as first come first served (FCFS), and shortest job first (SJF). Recently, reinforcement learning (RL) has also been studied and leveraged in various task scheduling problems. In [3], a DRL-based batch job scheduling method called RLScheduler is introduced. The RLScheduler learns the job requirements to satisfy their workloads and optimization goals. Moreover, the authors in [4] propose an RL-based scheduling method to reduce the waiting time of the jobs in the queue. Compared to the previous studies, this work focuses on the choice of an appropriate server to run a DL job among multiple options.

An earlier, conference version of this paper appeared in [5]. In this paper, we extend the previous paper by adding detailed explanations of monitoring an HPC-based AI computing environment and a DRL-based DL job scheduling scheme. The major contributions of this paper to the field are as follows. First, we proposed a multi-agent deep reinforcement learning (mDRL)-based resource allocation scheme for DL jobs to improve the system efficiency. Second, we conducted a more realistic performance evaluation in a real cluster computing environment.

The remainder of this paper is organized as follows. Sections 2 and 3 provide an overview of the background and related works on an HPC/AI converged system and resource allocation for DL jobs, respectively. Section 4 describes our system model (system description, problem definition), and the Section 5 provides the details of proposed resource allocation for DL jobs in the HPC/AI converged system. Section 6 demonstrates performance evaluations to validate the efficiency of the proposed resource allocation method for DL jobs. Finally, Section 7 summarizes the key findings and concludes the paper.

2 Background

2.1 Building HPC/AI Converged Cluster

As shown in Fig. 1, an HPC/AI converged cluster, called the GIST AI-X computing cluster, is developed for AI education and research in Gwangju Institute of Science and Technology (GIST).

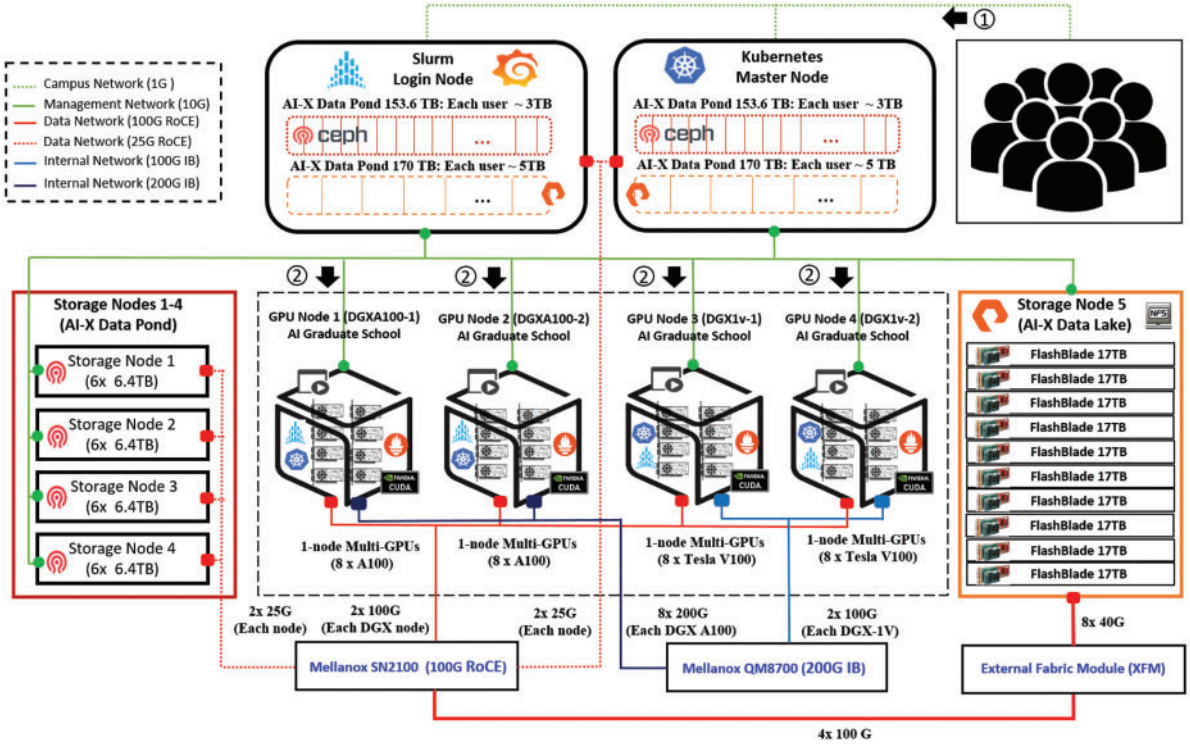

Figure 1: GIST AI-X computing cluster

Computing: NVIDIA DGX servers, with multiple GPUs with a large memory, are used to set the computing environment. The DGX servers are a line of NVIDIA-produced servers, which are configured to accelerate the performance of all ML/DL workloads. The DGX A100 consists of eight NVIDIA A100 GPUs with 320GB GPU memory, and the DGX-1 V consists of eight NVIDIA V100 GPUs 256GB of memory. Moreover, the DGX A100 and DGX-1 V servers achieve five and one petaFlops performances, respectively.

A combination of Prometheus and Grafana open-source tools is leveraged to monitor the GPU servers. The Prometheus is used to dynamically obtain the resource condition of each server, and the Grafana is used to visualize of the data that were collected through the Prometheus. In addition, users submit various jobs that require different versions of machine learning (DL) libraries, such as Pytorch and TensorFlow. These libraries require different CUDA versions depending on their version. For instance, a DL job that uses Pytorch 1.6 does not work on the DGX-A100 server, since the A100 GPU supports CUDA version 11 and Pytorch 1.6 requires CUDA version 10. Thus, it is essential to distinguish the DL jobs by their system requirement. In this regard, two partitions are created in the environment to satisfy the system requirement, such as A100 and V100 for DGX-A100 and DGX-1 V servers, respectively. For each partition, various types of computing resources are available for the DL jobs.

Storage: Two types of distributed storage solutions, such as network file systems (NFS) and Ceph, are efficiently leveraged to provide high-performance storage services to the systems. Moreover, nonvolatile memory express (NVMe)-based all-flash drives are used to set the storage services. NVMe is a storage access and transport protocol for nonvolatile memory-based storage that delivers the highest throughput and fastest response times for all types of workloads.

NFS is a file system that enables the storage and retrieval of data from multiple disks across a shared network. More specifically, NFS is implemented in a client/server computing mode, where it enables NFS clients to remotely access the data stored in the NFS server in the same way that they are accessed locally. NFS version 3 is used in our configuration, which provides a more efficient performance, safe asynchronous writes, and a smaller number of attribute requests. As shown in Fig. 1, storage node 5 (FlashBlade 170TB server) is used to set the NFS server. The other nodes (DGX servers, Slurm login and Kubernetes master nodes) functioned as NFS clients. In the GIST AI-X computing cluster, the NFS-based storage service is called the AI-X data lake, and 5TB storage is allocated to each user with this solution [6].

Ceph is an open-source, software-defined storage solution for data workloads. It is a distributed storage solution that provides highly scalable file, block, and object-based storage under a unified cluster system [7]. In addition, Ceph is fault-tolerant, and uses multiple disks over multiple servers to provide a single storage cluster with no single point of failure. In the GIST AI-X computing cluster, a Ceph-based storage service is called the AI-X data pond, and 3TB storage is allocated to each user with this solution. Moreover, the Kubernetes master node is simultaneously used as a Ceph master, and the storage nodes 1-4 are used to configure the Ceph dataponds.

Networking: As shown in Fig. 1, three networks, such as management, internal, and data, are built to configure the AI-computing cluster (colored green, blue, and red). The management network is configured to set the workload managers and connect the system to the Internet. The internal network is configured to enable multi-node training (connecting GPU nodes). The data network is used to provide a high-performance storage service to the whole system. Currently, network communication standards, such as Infiniband (IB) and remote direct memory access (RDMA) over converged Ethernet (RoCE), are commonly used as server and storage interconnects in the HPC environment. Thus, the internal and data networks in this setting are configured with IB and RoCE switches, respectively. IB is a high-performance communication standard, which provides a low latency and high bandwidth connection for system area networks (SAN) [8]. RoCE is a network protocol that enables RDMA over Ethernet network. RDMA provides direct memory access from the memory of one host to the memory of another host with low latency and high bandwidth [9].

2.2 Operating HPC/AI Converged Cluster

The GIST AI-X computing cluster is configured based on NVIDIA DeepOps, which encapsulates the best practices for deploying GPU server clusters and sharing computing resources for multiple users [10]. There are three deployment options for the DeepOps, such as Slurm-based, Kubernetes-based, and hybrid, which supports both the Slurm and Kubernetes. As shown in Fig. 1, the hybrid configuration is used to set the GIST AI-X computing cluster. Slurm is an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system installed in about 60% of the top supercomputers in the world. As a cluster workload manager, Slurm is responsible for efficiently distributing the computing resources among the users and mapping the submitted jobs to the underlying hardware. It has three major functions, such as resource management, job management, and a scheduler. First, the resource manager allocates either exclusive or nonexclusive access to the

computing resources to users for some duration. Second, the job manager provides a framework for starting, executing, and monitoring jobs on the set of allocated compute nodes. Finally, the scheduler arbitrates contention regarding the computing resources by managing a queue of pending works [11].

Running DL Jobs: Frameworks, such as Tensorflow and Pytorch are essential for implementing DL applications. They provide a collection of workflows to develop and train neural networks. Since the GIST AI-X computing cluster is a shared system, installing all required libraries of multiple users in the system could be a reason for the various compatibility issues. In this regard, containerization technology is adapted to independently provide all user requirements. Users can use any docker container that includes their desired software libraries and execute their job inside the container. However, the docker container should be converted to a singularity container to run in the Slurm environment. Singularity is an HPC-optimized containerization technology, which enables users to have full control of their environment. This means that users do not need to ask the cluster-admin to execute specific containers in the system. Instead, they can download and run any singularity container in the system by themselves. The singularity can import docker images without a docker being installed in the system or requiring a superuser [12].

Distributed Training: Deep neural networks (DNN) with a large number of parameters require large amounts of data to learn their model. This is a computationally intensive process, which takes a lot of time. Thus, it is essential to offer parallel and distributed training, which can run faster and reduce the training time.

In this study, parallel computing for distributed training is enabled using OpenMPI. The OpenMPI is an open-source message passing interface (MPI), where the MPI is a distributed communication standard for parallel computing. MPIs in OpenMPI are used to invoke processes and send/receive messages among different processes running on the same or different machines [13].

Scaling DL with distributed training considerably reduces the training time and improves the research progress. It encounters some obstacles, such as the training model needing to support inter-GPU communication and users needing to modify their training code to run with multiple GPUs. Thus, Uber developed a framework called Horovod to make the distributed DL fast and easy to use. Horovod takes a single GPU training script as an input and successfully scales it to train across multiple GPUs in parallel [14]. In the GIST AI-X computing cluster, users mostly employ the Horovod container to use multiple GPUs for distributed training.

3 Related Work

3.1 HPC/AI Converged Clusters

To emphasize diversified approaches and efforts for building HPC/AI clusters, several works are summarized in Tab. 1. In fact, various sizes of clusters have been built depending on the main contributors and target users. For example, a laboratory could build a toy example of clusters with around 10 servers, and university consortium could build multiple clusters with around 100 servers. And national institutes backed by world-leading companies and government have been building hyper-scale supercomputer clusters. To show such great interest in various scales, we arbitrary defined the size of the clusters.

Table 1: High-performance AI computing clusters

Scale	Scale	Spec. (in total)	Notice
Hyper	(a) SUMMIT [15]	■ 4608 nodes of CPU/GPU hybrid system ■ 202,752 CPU cores of IBM power 9 processors ■ 10.2PB memory of HBM and DRAM ■ 27648 NVIDIA V100 GPUs with NVLink ■ 250PB IBM GPFS-based storage pool ■ 2 x 100G NVIDIA infiniband EDR	Peak performance: ■ 200 PFLOPS
	(b) SIERRA [16]	■ 4320 nodes of CPU/GPU hybrid system ■ 190,090 CPU cores of IBM power 9 processors ■ 17280 NVIDIA V100 GPUs with NVLink ■ 1.29PB RAM ■ 2 x 100G NVIDIA infiniband EDR ■ 154 PB IBM GPFS-based storage pool	Peak performance: ■ 125 PFLOPS
	(c) Sunway Taihulight [17]	■ 40960 nodes of many-core CPU system ■ 10 million cores of home-grown CPU processor ■ 1310.72TB RAM ■ 20PB storage Pool ■ 16G network link bandwidth	Peak performance: ■ 125 PFLOPS
	(d) Milkyway-2 (Tianhe-2) [18]	■ Compute subsystem: 16000 compute nodes ■ 384,000 cores of Intel Xeon E5-2600 v2 Processors, and 2,736,000 cores of Intel Xeon Phi Co-processors ■ 1.024PB RAM for CPU and 0.384PB RAM	Peak performance: 54.9 PFLOPS

(Continued)

Table 1: Continued

Scale	Scale	Spec. (in total)	Notice
Middle	(a) Hybrilit [19]	■ Heterogeneous computing clusters ■ HybriLit education and testing polygon: 10 nodes with Intel Xeon processors and NVIDIA GPUs ■ GOVORUN supercomputer: 40 nodes with Intel Xeon processors, 21 nodes with Intel Xeon Phi many-core processors, 5 NVIDIA DGX-1 V nodes. ■ Intel Omni-path Inter-connect ■ SLURM-based ML/DL workload orchestration	Peak performance: ■ Testing polygon 142 TFLOPS ■ GOVORUN 1000 TFLOPS
	(b) Cluster-UY [20]	■ 28 computing nodes ■ 2 storage node ■ 1 service (DevOps) node ■ 560 cores of Intel Xeon Gold CPUs ■ 28x NVIDIA P100 GPUs ■ 3.584TB RAM ■ 90TB external storage pool ■ 1G for management and monitoring ■ 10G for control and data traffic	Peak performance: ■ 166 PFLOPS
	(c) OCCAM [21]	■ 32 light nodes (2x Intel E5-2680 CPUs, ■ 128GB RAM, 400GB SSD) ■ 4 fat nodes (4x Intel E7-4830 CPUs 768GB RAM, 800GB SSD + 2TB HDD) ■ 4 GPU nodes (2x Intel E5-2680 CPUs, 128GB RAM2x NVIDIA K40 GPUs, 800GB SSD)	■ Deploying workloads with Docker containers ■ Creating virtual clusters by utilizing Ansible and Marathon

(Continued)

Table 1: Continued

Scale	Scale	Spec. (in total)	Notice
		■ Multi-tiered storage pool: 256 TB ■ 768 TB NFS-enabled archive pool. ■ 1G control/management and 10 G data networks ■ 56G NVIDIA Infiniband FDR	
Small	(a) CALIBAN [22]	■ 17 nodes of CPU/GPU hybrid system ■ 544 cores of 68 AMD CPUs ■ 6x NVIDIA GTX 670 GPUs ■ 1.01TB RAM ■ 7.446TB Local HDD Storage Pool ■ Mellanox Infiniband 40G ■ 1G network for SAN ■ 50TB external storage pool	■ Parallel Computing ■ MPI and openMP
	(b) EVOLVE [23]	■ Cluster with heterogeneous computing resources (CPU, GPU, and FPGAs) ■ 10 compute nodes (each with 24 cores and 128GB RAM) ■ 5 accelerator nodes. ■ NVIDIA InfiniBand FDR Link (56GB/s) under a fat-tree fabric interconnect ■ 128TB Lustre filesystem	■ Kubernetes-based ■ cloud-native computing ■ Custom schedulers ■ Unified Storage Layer ■ Monitoring services
	(c) SmartX Intelligence Cluster [24]	■ 5 computing nodes with Intel Xeon CPUs and Nvidia TitanV GPUs ■ 400TB all-flash storage nodes ■ RoCE-based 100G network	■ Kubernetes-based cloud-native computing

Notes: Hyper scale: More than 1000 nodes/Middle scale: 20-1000 nodes/Small scale: Less than 20 nodes
The detailed specifications can be changed over time. This table shows the specification described in the referenced manuscripts.

Due to the complexity of DL models and large training data, a demand of scaled computing for AI research and development is continuously increasing. While existing systems provide computing resources through networks for end users, the current solutions still face many challenges. The

end devices could be self-driving cars, drones, robots that require data processing and decision making closer to the point of data generation due to mission critical, low-latency and near-real time requirements. In this regard, most of the organizations desire to build their own scaled AI computing cluster close to their development environment. However, building and operating the scaled computing environment for AI is not easy task, since it requires broad knowledge of networked systems and high cost. In this study, we introduce a case of building and operating an AI computing environment suitable for small or middle-sized research and educational institutes.

3.2 Job Scheduling and Resource Allocation

There are several existing works on resource allocation and job scheduling in HPC/AI converged systems [25–35]. Early studies examine the selection of a priority for the batch jobs. They developed various scheduling policies, such as first come first served (FCFS), and shortest job first (SJF). Recently, reinforcement learning (RL) was studied and leveraged in various task scheduling problems.

Thinakaran et al. [25] proposed Kube-Knots for multi-node GPU utilization-aware orchestration along with QoS-aware container scheduling policies. The Kube-Knots are leveraged by the Kubernetes at the cluster level to monitor the real-time GPU utilization at every node. They also proposed Peak Prediction (PP) and Correlation Based Prediction (CBP) schedulers that perform safe co-locations through GPU spare cycle harvesting by dynamically resizing the containers (i.e., crash-free dynamic container resizing). It also performs QoS-aware container co-locations on GPUs at the cluster-level without a priori knowledge of incoming applications. In Shao et al. [26], proposed a GPU scheduling method using a modified shortest-job-first scheduling policy with respect to GPU-sharing for short GPU tasks of multiple users. The scheduling method is implemented by a container-based batch computing system that accepts and runs users' jobs through container images with specified configurations, without the user having to care about resource leases and releases. The proposed method ensures the priority of the short tasks and prevents long tasks from starving. Chen et al. [27] proposed a QoS-aware dynamic scheduling framework for a shared GPU cluster to achieve users' QoS guarantee and high system utilization. They proposed a predictive model derived from lightweight profiling to estimate the processing speed and response latency for various Deep Learning (DL) workloads. The QoS-aware scheduling algorithm identifies the best placements for DL tasks and schedules them on the shared cluster. They showed that the proposed method on real clusters and simulations achieves higher QoS guarantees and system utilization than other reference methods. In Filippini et al. [28], proposed a hierarchical approach coupled with a novel Mixed Integer Linear Programming (MILP) formulation to overcome limitations of scalability issues with respect to problem instances with a small number of nodes and jobs. Their proposed method optimizes operating costs by rightsizing VM capacity on each node, partitioning GPU sets among multiple concurrent jobs on the same VM, and scheduling deadline-aware jobs. They showed a good fit with real-world scenarios, as the scalability analysis showed that the proposed method can solve problem instances with up to 100 nodes in less than one minute on average.

In Habuza et al. [29], proposed a user-friendly jobs and resources allocation manager (i.e., web-based multi-user concurrent job scheduling system) for the Machine Learning (ML) server. The proposed manager helps users to request and allocate resources from the server and monitor the progress of their jobs to provide AI research or computations with an optimized combination of compute power, software and deep learning performance. There are three main task manager components: LoginServer, ML server (DGX-1) and Data Storage. Luan et al. [30] proposed a GPU cluster scheduler using deep reinforcement learning (DRL) to execute smart locality-aware scheduling for efficient deep learning training (DLT) jobs. The proposed scheduler controls fragmentation in GPU

clusters by reflecting the current and future degree of cluster fragmentation. It also improves cluster utilization and job performance by using the profiling data to reflect job's locality-sensitivity. They show that SCHED2 can effectively control cluster fragmentation and job locality, reduce the JCT by 4.6 times, and make the span 2.1 times. In Wang et al. [31], proposed a non-intrusive GPU scheduling framework by integrating an adaptive GPU scheduler and an elastic GPU allocation method to reduce the span and improve resource utilization. The GPU scheduler can determine the most efficient allocation and reallocation of GPUs by using training job progress information for incoming and running jobs at any given time. The GPU allocation method temporarily stops and restarts the job's DL training process with a different number of GPUs based on a "SideCar" process to further reduce the reshaping overhead. The proposed method has been shown to reduce overall execution time and average job completion time by up to 45% and 63%, respectively, compared to the default scheduler. In Abdulla et al. [32], proposed a deep learning-based method to automatically allocate optimal CPU resources to the containers. The proposed method decides on the optimal number of CPU pins using the law of diminishing marginal returns for the maximum performance of containers while maximizing the number of concurrent jobs. They showed the effectiveness of the proposed method in reducing the completion time of the jobs to compared to static CPU allocation methods (i.e., First Come First Serve (FCFS), Shortest Job First (SJF), Longest Job First (LJF), and Simulated Annealing (SA)).

Zhao et al. [33] proposed a scheduling system (i.e., CODA) that enhances the resource utilization of GPU clusters. The proposed system consists of an adaptive CPU allocator, a real-time contention eliminator, and a multi-array job scheduler. CODA uses the feedback-based technique to identify a sufficient number of cores for a DNN train job. CODA removes interference to improve the performance of jobs by analyzing the competition for CPU resources in the performance of DNN training jobs. CODA's multi-array job scheduler removes GPU fragmentation. Their system improves GPU utilization by 20.8% on average without increasing the queuing time of CPU jobs. In Lin et al. [34], proposed multiple intelligent schedulers based on a two-stage job scheduling and resource allocation framework for the cooperative scheduling problem of job scheduling and resource allocation regarding multi-user multi-cloud data centers. They used the heterogeneous distributed deep learning (HDDL) model to schedule multiple jobs and the deep Q-network (DQN) model to deploy virtual machines, respectively. The proposed method, using an HDDL-based task scheduler and DQN-based resource allocator in numerical simulation experiments, can achieve a better performance and computation delay than the baseline.

Existing works on resource allocation for HPC systems are broadly categorized into two main parts such as energy efficiency and performance-based. Energy efficiency-based works try to minimize a number of running servers, and the performance-based methods try to complete submitted jobs as fast as possible. The energy efficiency-based method is similar to the default scheme (Naïve), which finds an available server from the first server in order to minimize the number of running servers. Performance-based models try to allocate computing resources as much as possible for a submitted job. However, our work allocates fixed resources for each job based on the allocated flavor type.

4 System Model

4.1 System Description

As illustrated in Fig. 2, we consider a high-performance AI computing cluster environment. The AI computing cluster architecture is composed of a login, compute, and storage nodes. The login node is a CPU server, which monitors the resource utilization of the computation nodes and allocates computing resources for user jobs. The compute nodes are GPU servers that user jobs originally

run with the requested computing resources. In brief, a user job is submitted to one of the GPU servers when the server has enough resources to run the job. The GPU servers are grouped into multiple partitions depending on the user and system requirements. For instance, each department in a university can use a different partition. Moreover, the GPU servers can be categorized into different partitions depending on their system specifications. Additionally, the login, compute and storage nodes are connected through management, internal, and data networks. The management network is used to control and manage the resource management system among the cluster. The internal network is high-speed Infiniband network, which is used to connect the GPU servers to enable multi-node training and parallel computing. Lastly, the data network is high-speed RoCE network, which is used to connect GPU servers to the storage node.

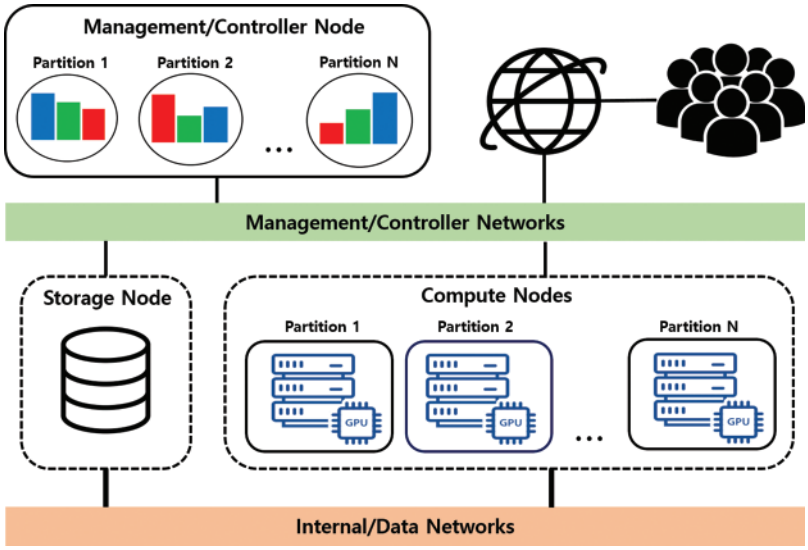

Figure 2: System description

In order to run a job on the GPU nodes, users first connect to the login server; then, they request a certain amount of computing resources from a dedicated partition. The user job will automatically start to run with the requested resources if there are enough computing resources in the partition. In addition, the job will be in the pending state if there are not enough resources in the partition. The maximum amount of computing resources that a user can request can be configured in the system. In cloud and cluster computing, the set of computing resources that can be allocated to each user is generally called a flavor. As shown in Tab. 2, an s-A100 type job can use, at most, two A100 GPUs, 64 CPU cores, and 256GB memory. Moreover, the time limit for running a s-A100 type job is 72 h.

4.2 Problem Definition

We assume there are N GPU servers that are grouped into M partitions depending on their system specifications and user requirements. Let $P = \{p_1, p_2 \ldots p_N\}$ be a set of all partitions in the cluster. Then, the total number of GPU servers is

$$N = \sum_{i=1}^{M} |p_i| \tag{1}$$

where $|p_i|$ is a number of GPU servers in i-th partition.

Table 2: Resource Allocation Policy (Flavor)

Name	GPU (number)	CPU (cores)	Memory (GB)	Time (t)	Rate (λ)
s-A100	2× A100	64	256	72 h	0.125
m-A100	4× A100	128	512	168 h	0.25
l-A100	8× A100	256	1024	336 h	0.5
f-A100	16× A100	512	2048	504 h	1
s-V100	2× V100	20	128	72	0.125
m-V100	4× V100	40	256	168 h	0.25
l-V100	8× V100	80	512	336 h	0.5
f-V100	16× V100	160	1024	504 h	1

Let $d_{i,j}$ be j-th DL job, submitted to an i-th partition. As mentioned in 4.1, one of the flavors (resource allocation type) is allocated to each DL job. The flavors could be as same as illustrated in Tab. 2. Then, $\lambda(d_{i,j})$ is a function used to find a resource allocation rate for $d_{i,j}$ among all the resources in a partition p_i, where $0 < \lambda \leq 1$. For instance, if a flavor s-A100 in Tab. 2 is allocated for a job $d_{i,j}$, its resource rate $\lambda(d_{i,j})$ is 0.125. Let $t(d_{i,j})$ be the function used to check the configured time limit for the job $d_{i,j}$. For instance, $t(d_{i,j})$ can be 72 h if a flavor s-A100 is allocated to the $d_{i,j}$. Then, $\tau(d_{i,j})$ is the total run time of the job $d_{i,j}$ in the system, and it is computed as

$$\tau(d_{i,j}) = \begin{cases} t_c - d_{i,j}.t_s, & \textit{if running} \\ t(d_{i,j}), & \textit{if completed} \end{cases} \tag{2}$$

where $d_{i,j}.\ t_s$ is a start time of the job $d_{i,j}$ and t_c is current time. Moreover, let $T_{t_c}^{p_i}$ be the total run time of the partition p_i at time t_c. We consider a partition run time to have started when the first job is submitted. Then, it can be computed as follows.

$$T_{t_c}^{p_i} = t_c - \min_{d_{i,j} \in p_i} \{d_{i,j}.t_s\} \tag{3}$$

Then, an utilization rate U of p_i at time t_c can be computed as follows.

$$U_{t_c}^{p_i} = \frac{\sum_{d_{i,j} \in p_i} \lambda(d_{i,j}) \cdot \tau(d_{i,j})}{R \cdot T_{t_c}^{p_i}} \tag{4}$$

R is a total resource rate of each partition, which it is generally set as 1.

Fig. 3 shows two cases where five DL jobs are scheduled in a single partition. In this example, we assume the partition is A-100. Let a rectangle be a job submitted to the system, where its width is the ratio of required resources and height is the time taken to complete the job. As shown in Tab. 2, the flavors s-A100, m-A100, m-A100, l-A100, and f-A100 are allocated to jobs 1-5, respectively. Consider the jobs submitted sequentially to the system at times t_0, t_1, t_2, t_3, and t_4.

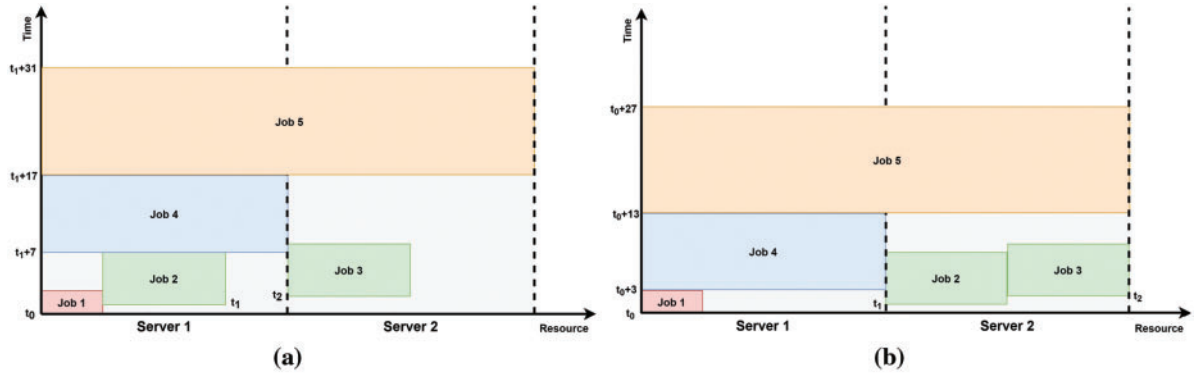

Figure 3: Resource allocation and scheduling cases (a) default (b) proposed

In case 1, the system used the default resource allocation rule that checks the available resources for the job from the first server. Then, it allocates resources from the first discovered server for the job. As shown in case 1, jobs 1 and 2 are submitted to server 1, and job 3 is submitted to server 2, since there is an insufficient resource for job 3 in server 1 at time t_2. At time t_3, there is an insufficient resource for the job 4. Thus, the job is submitted at time $t_1 + 7$, when server 1 has resources. Then, job 5 is submitted to the system at time $t_1 + 17$, when both servers are released.

At time t_1, an agent has two options for running job 2, which the job 2 can be submitted to either server 1 or 2. Following the default resource allocation rule, job 2 is submitted to server 1 in case 1, since there are enough resources to run the job. In case 2, job 2 is submitted to server 2. As a result, submitting job 2 to server 2 helped to complete all the jobs at time $t_0 + 27$ in case 2, while it is $t_1 + 31$ in case 1. Following the run time in Tab. 2, all jobs are completed at least 4 days earlier in case 1, compared to case 2. Besides this, inefficient utilization of the system, gray space in the figure, is considerably high in case 1 compared to case 2. Thus, an efficient resource allocation scheme is essential to improve the efficiency of monitoring the HPC/AI converged system.

In this regard, we aim to improve the total system utilization. Thus, the proposed resource allocation for monitoring the AI computing environment is as follows.

$$
\begin{aligned}
X^* = \arg \max_X \sum_{p_i \in P} U_{t_c}^{p_i} \\
\text{subject to } \sum_{d_{i,j} \in p_i} d_{i,j}.g \leq p_i.g \\
\sum_{d_{i,j} \in p_i} d_{i,j}.c \leq p_i.c \quad \text{for } \forall i \in [1,\ \mathrm{M}] \\
\sum_{d_{i,j} \in p_i} d_{i,j}.m \leq p_i.m
\end{aligned}
\tag{5}
$$

The $d_{i,j}.g$, $d_{i,j}.c$, and $d_{i,j}.m$ are the respective allocated amounts of GPU, CPU, and memory resources to the $d_{i,j}$. Moreover, t $p_i.g$, $p_i.c$, and $p_i.m$ are the respective total amounts of GPU, CPU, and memory resources in partition p_i.

5 Proposed Resource Allocation

5.1 Multi-Agent Deep Reinforcement Learning

At present, many sequential decision-making problems are solved using reinforcement learning (RL). In RL, an agent interacts with the environment and tries to learn an optimal policy that maximizes the utility of its actions. At each time step, the RL agent predicts the best action to execute in the current state and executes it. To predict the best agent action, an agent selects the action with maximum cumulative reward from all possible actions in the current state. The cumulative reward is referred to as the quality value (Q value). An agent updates a Q value for each executed action based on the immediate reward received from the environment. The Q value for an action in state s_t is calculated using Eq. (6), called the Bellman equation.

$$Q(s_t,\ a_t) = r_t + \gamma \cdot \max(Q^*(s_{t+1},\ a_{t+1})|s_t,\ a_t) \tag{6}$$

The immediate reward r_t can be obtained directly from the environment immediately after the action a_t. The Q value for the next state s_{t+1} is calculated based on either the Q table or approximated with DNN. The γ is a fixed value (discount factor), which controls the importance of long-term rewards *vs.* the immediate reward. Moreover, memorizing all states in the Q table is unfeasible when the state and action space is large. In this regard, DNN can be used to approximate expected Q values for future states. The RL that uses DNN to model the policy is called deep reinforcement learning (DRL) [3].

Most real-world problems, such as autonomous driving and robotics, require multiple policies to control their tasks, even though the tasks share the same environment. In this regard, multi-agent DRL (mDRL) is used to solve sequential decision-making problems that require multiple policies. This is the same as single-agent RL, where each agent tries to learn their solution [3].

5.2 Proposed Strategy

As illustrated in Fig. 4 and Algorithm 1, mDRL is employed to allocate resources for DL jobs in the AI computing environment. The mDRL methods for the proposed solution are defined as follows.

Agents: A single centralized agent can observe the entire system state and decide actions for all partitions. In practical implementations, it is not easy for the agent to find an optimal solution because the action space increases significantly as the number of partitions increases. Thus, we employ the decentralized mDRL and consider the job scheduling policy for each partition as an agent. Once a job arrives in the system, an agent is selected based on the job partition. Then, the agent decides the best action for the job, which has the largest Q value among the possible actions in the current state. In each agent, DNN is used to approximate Q values for all possible actions. Initially, Q values are set to 0 in all states. They are updated while the agents interact with the environment. To train the DNN, an agent performs a certain number of actions by providing them with positive rewards for the right action and trains them to stay away from others by providing negative rewards.

State: The state-space of the AI-computing cluster for DL job scheduling is a set of available resources in each GPU server. With the monitoring tools in the AI-computing cluster, available resources in each server can be obtained dynamically. Thus, each agent can obtain the state information of its dedicated partition. One possible state-space is shown in Fig. 4. As shown in the figure, full resources are available in all DGX servers. The resource information of the servers can be an initial state. Besides this, an episode in RL is the same as taking actions from the initial to the terminal state. In this study, the initial state for each partition indicates no running job in the partition. The terminal state is a state after submitting a job that uses all the partition's resources. For instance, submitting an f-A100 type job for partition A100 and f-V100 type job for partition V100 can serve as terminal states.

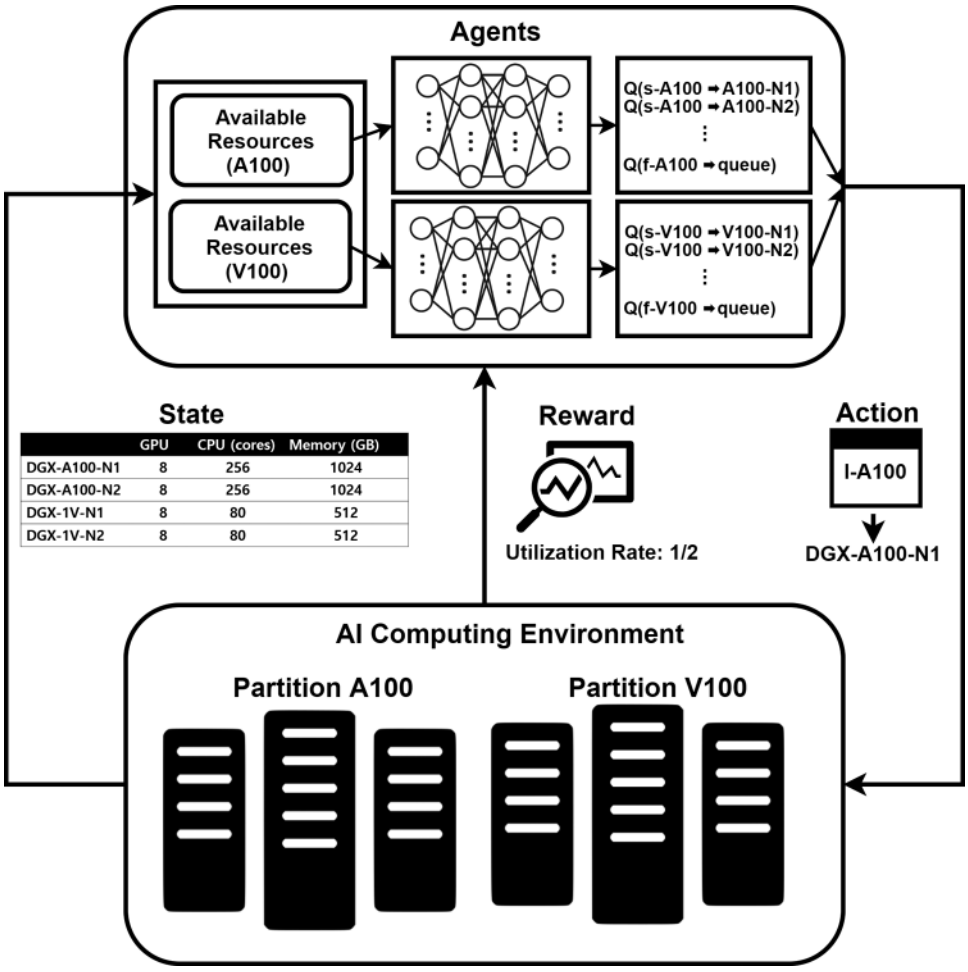

Figure 4: Proposed strategy

Action: Since the environment is an AI-computing cluster, the action space is a set of possible actions for a DL job, such as running the job in either single or multiple GPU servers and putting the job in the queue when there are insufficient resources to run the job. In this study, DL jobs are categorized based on their allocated resource type. Thus, the combination of a flavor and GPU server is considered as the action of running a job. Besides this, putting each different type of job in the queue is considered an action. For instance, let n be the number of available flavors that can be allocated in a single server and m be the number of GPU servers in the i-th partition. Thus, the number of possible actions to run all types of jobs in a single server is $n \cdot m$, and there are n additional actions to put the jobs in a queue (putting different type of jobs in queue are considered different actions).

Actions in previous studies are mostly considered to decide on the priority of the jobs in the queue. In practical, critical jobs, such as paper experiments and training DNN for real applications, need more computing resources and time. However, non-critical jobs, mostly submitted from new users, such as testing the GPU cluster or practicing DL with multi-GPUs, do not require many resources or a long time. In this study, the job priority is set based on the largest job-first strategy (LJF). For instance, the highest priority for the full type of flavor job and the lowest priority for the small flavor type of job.

Reward: If a job is added to the queue when there are enough resources to run the job in the dedicated partition, its reward is −1. Moreover, the reward is −1 when the selected action for a job is to execute it in one of the servers, while there are insufficient resources in the partition. Here, the job is added to the queue. If there is an insufficient resource in all possible servers and the selected action for a job is putting it in the queue, its reward is 0. If the selected action for a job is executing the job in one of the GPU servers, and there are enough resources to run it, its reward is the utilization rate of the dedicated partition. The utilization rate for partition p_i at time t is computed with Eq. (4).

Algorithm 1 Resource allocation algorithm for DL jobs

Input:
A buffer for receiving jobs J
Number of episodes E
Number of jobs K
Initialize:
Set agent A_i for each partition p_i, $\in [1, M]$
Set a replay memory$D$
Set a policy network Q_i^{ω}, random weights ω_i and θ_i
Set a target network Q_i^{ω}, weights $\omega_i' \leftarrow \omega_i$, $\theta_i' \leftarrow \theta_i$,
Set a replay memory D
while J is not empty **do**
    for episode = 1 to E **do**
        while not terminal state **do**
            Get a job $d_{i,j} \leftarrow J, j \in [1, K]$
            Select an agent $A_i \leftarrow d_{i,j}$
            Check state $si \leftarrow A_i, P_i$
            Select action $a^i \leftarrow Q_i, D_{i,j}$
            Submit a job $p_i \leftarrow d_{i,j}$
            Set a start time $d_{i,j}.s \leftarrow t$
            Execute a^i and observe $ri \leftarrow U(p_i, t)$
            Update $D \leftarrow ai_t, ri_t, s^i{}_t, s^i{}_t+1$
            Update $Q_i(si, a^i) \leftarrow ri, Q_j(si, a^i)$
        end while
    end for
end while

6 Performance Evaluation

6.1 Simulations

We evaluated the performance of the proposed DL job-scheduling scheme with numerous simulations. The simulation environment is configured similarly to the case introduced in Fig. 3. To demonstrate the proposed strategy's effectiveness, we trained the DNNs for each agent with 1000 episodes, with each episode consisting of five DL jobs. The flavors allocated to the jobs are randomly selected from the flavors in the Tab. 2. An episode is considered as the process of submitting a number of DL jobs to the cluster until the terminal state. The terminal state is the state after submitting a job that uses all the partition's resources. For instance, the terminal state job can be either an f-A100 or f-V100 job, as introduced in Tab. 2. Moreover, one of the s-A100, m-A100, and l-A100 flavors can be

allocated to the jobs that run in each episode if the partition is A100. Similarly, one of the s-V100, m-V100, and l-V100 flavors can be allocated to the jobs that run in each episode if the partition is V100. In addition, the performance of the proposed method is compared with a Naïve scheme that submits a job to a randomly selected GPU server if the server has enough resources to run the job.

Fig. 5 shows the results of the system utilization rate with respect to the number of episodes. The results of both the proposed and Naïve schemes are illustrated in Figs. 5a and 5b, respectively. Initially, the utilization rate with the proposed strategy fluctuates in the long range, but it converges after 300 episodes, with its average utilization rate converging to more than 0.75. However, the utilization rate of the Naïve method continuously fluctuates in the long range, and frequently drops to less than 0.3. The reason for the lower utilization of the Naïve method is that it does not consider future rewards, while the proposed scheme continuously learns the model and considers the current state when deciding on an action. The simulation results indicate that the use of proposed strategy considerably improves the efficiency of the system utilization compared to the Naïve scheme.

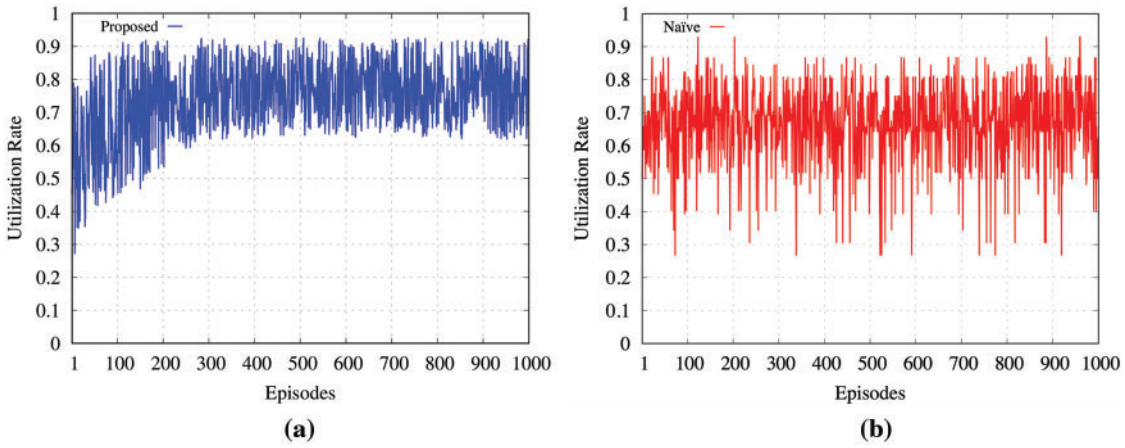

Figure 5: Utilization rate with respect to episodes (a) utilization rate for proposed allocation (b) utilization rate for default allocation

6.2 Experiments

To conduct a more realistic performance evaluation, we carried out series of experiments in GIST AI-X computing cluster. As introduced in Section 2.1, two DGX-A100 and two DGX-1 V GPU servers are used to set the compute nodes. DGX A100 and 1 V servers have eight NVIDIA ampere A100 GPUs with 320GB GPU memory and eight NVIDIA V100 GPUs with a total of 256GB of memory, respectively. Additionally, two SuperServer 1020U are used to set the Slurm login and Kubernetes master nodes. Each SuperServer has dual 6-core Intel Xeon CPUs, 64GB system memory, 4TB NVMe SSD, two 25G, and two 10G NICs. Additionally, two storage servers, such as Purte storage Flash Blade (170 TB) and SuperMicro 2u4N (Total 153.6TB), are used to set the storage services.

Moreover, Mellanox QM8700 200G (IB), Mellanox SN2100 100G (RoCE), and Netgear 10G (Ethernet) switches are used to configure the internal, data, and management networks. Eight 200G ports of each DGX-A100 and two 100G ports of each DGX-1 V are used to set the internal network. Since the IB switch used for the internal network has 200G ports (HDR200), two of them are splitted into dual 100G ports for DGX-1 V servers. Two 100G ports of each DGX- 1 V are used to connect the server to the data network. Since the other nodes (Slurm login, Kubernetes Master, and the storage

nodes 1-4) have only dual 25G NICs, one 100G port in the RoCE switch are splitted into four 25G to connect the 25G nodes to the data network. Due to the 40G NICs in storage node 5 (data lake node), the external fabric module (XFM) is used to connect the server to the 100G data network.

With the trained a DNN for each agent, 100 episodes are randomly created in order to test the efficiency. Each episode consists of five DL jobs, where four jobs can be run in a single server, and the last job is a terminate state job, which uses all the partition's resources, is submitted. For the jobs in each episode, flavors s-A100, m-A100, l-A100, f-A100 in Tab. 2 are allocated for the partition A100 and flavors s-V100, m-V100, l-V100, f-V100 in Tab. 2 are allocated for the partition V100. To shorten the experiment time, the time limit is set as minutes, which is defined as hours in Tab. 2.

In Fig. 6a, the blue lines show the results of the system utilization with the proposed strategy in partition A100, and the red dashed line illustrates the average resource utilization rate with the Naïve method. Additionally, the performance of the resource utilization with the proposed method in partition V100 is depicted in Fig. 6b. In both partitions, the average utilization rate of the Naïve scheme is lower than the system utilization of the proposed strategy in all the episodes. As shown in the figure, the average system utilization with the proposed scheme is more than 0.75, while it is around 0.62 with the Naïve method. Moreover, the total run time of the system in each episode is reduced by around 20% with the proposed scheme.

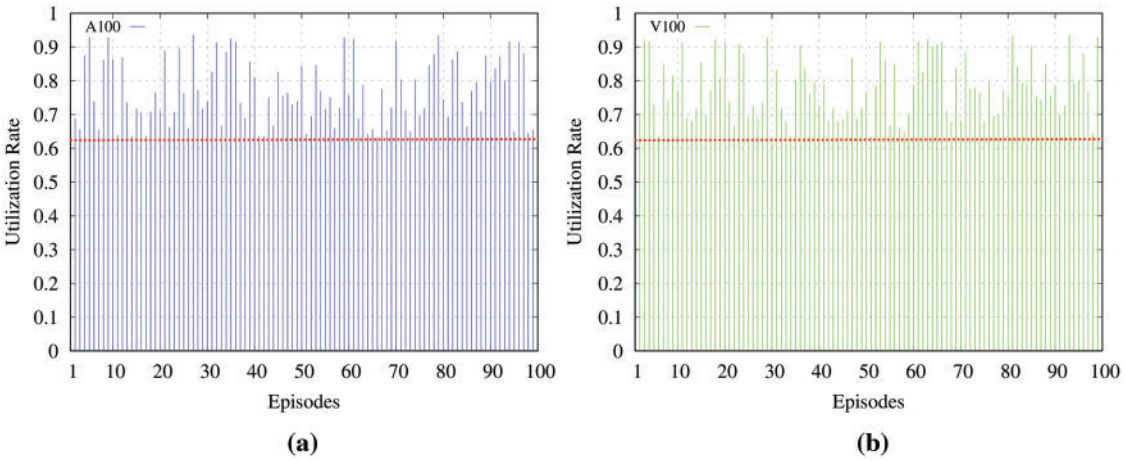

Figure 6: Experiment results (a) utilization rate (A100) (b) utilization Rate (V100)

Additionally, power consumption in HPC cluster is one of the main measurements. Each DGX-A100 servers uses 6.5 KW power, and each DGX-V100 server uses 3.5 KW power. In the GIST AI-X cluster, there are two DGX-A100 servers in A100 partition and two DGX-V100 servers in V100 partition. Thus, power consumption of the A100 and V100 partitions are at least 13 KW and 7 KW, respectively. The power consumption analysis is depicted on the Fig. 7. If we consider an episode runs for 720 h, the system runs for 273.6 h (720 x 0.38) inefficiently with the default resource allocation method. Then, inefficient power consumption for partition A100 is 3.55 MW (273.6 kW x 13 KW), and 1.91 MW for partition V100. With the proposed method, it is decreased to 2.15 MW and 1.15 MW for A100 and V100 partitions, respectively. Thus, inefficient power consumption is reduced around 40% with the proposed method.

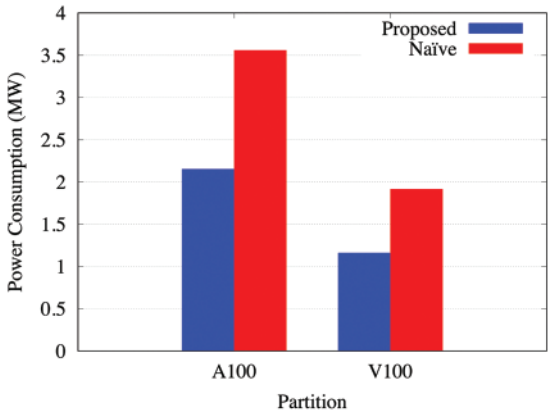

Figure 7: Power consumption analysis

The reason for the improvement is that the proposed method considers future reward by leveraging the feature of reinforcement learning. However, existing works only consider the current state and find available servers to allocate resources for jobs from first to end.

7 Conclusion and Future Work

In this study, we introduce a method of building and operating an HPC/AI converged environment, which is suitable for small or middle-sized research and educational institutes. Additionally, we propose a DL job-scheduling method with mDRL, which considerably improves the efficiency of system utilization. Through extensive simulations and experiments, we validate that the proposed mDRL algorithm can help the HPC/AI converged cluster to achieve both system utilization and power consumption improvement. By deploying the proposed resource allocation method, total job completion time is reduced by around 20% and inefficient power consumption is reduced by around 40%. In the future, we plan to collaborate with other organizations who have HPC/AI computing clusters, then we will evaluate a performance of the proposed method on other systems.

Moreover, running AI applications on resource-constrained devices encounters severe challenges. In this regard, HPC/AI converged system is becoming a major trend for delivering infrastructure on demand via the network. While existing systems provide computing resources through networks for end users, the current solutions still face many challenges. The end devices could be self-driving cars, drones, robots that require data processing and decision making closer to the point of data generation due to mission critical, low-latency and near-real time requirements. However, the traditional centralized HPC-clusters fail to meet the requirement, since it is usually located far from the end users. To satisfy the requirement of low-latency and near-real time data processing, HPC/AI clusters should be built close to the source of data generation with edge computing and only send filtered data to the centralized system when it requires large storage or higher-level analysis.

Acknowledgement: This work was supported by Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2019-0-01842, Artificial Intelligence Graduate School Program (GIST)). We would like to thank the Cloud team at Kakao Enterprise, as well as especially thank Jung-Bok Lee for his support during the paper revision.

Funding Statement: This research was funded by Artificial Intelligence Graduate School Program (GIST).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] NVIDA, "NVIDIA DGX SuperPOD: Scalable infrastructure for AI leadership.," 2020. [Online]. Available: https://images.nvidia.com/aem-dam/Solutions/Data-Center/gated-resources/nvidia-dgx-superpod-a100.pdf.

[2] IBM and MIT, "Satori." 2021. [Online]. Available: https://mit-satori.github.io/.

[3] D. Zhang, D. Dai, Y. He, F. S. Bao and B. Xie, "RLScheduler: An automated HPC batch job scheduler using reinforcement learning," in *SC20: Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Online conference, pp. 1–15, 2020.

[4] S. Liang, Z. Yang, F. Jin and Y. Chen, "Data centers job scheduling with deep reinforcement learning," in *Advances in Knowledge Discovery and Data Mining*, vol. 12085, pp. 906, 2020.

[5] N. Jargalsaikhan, S. Jun-Sik, P. Sun and K. JongWon, "Optimizing computing environment for scaling deep learnig applications," in *Int. Conf. on Advanced Engineering and ICT-Convergence (ICAEIC)*, Online conference, 2021.

[6] U. Rawat and S. Roy, "An efficient technique to access cryptographic file system over network file system," in *Innovations in Computational Intelligence and Computer Vision*, vol. 1189, pp. 463, 2020.

[7] J. LeFevre and C. Maltzahn, "SkyhookDM: Data processing in ceph with programmable storage," *USENIX Login*, vol. 45, no. 2, 2020. [Online]. Available: https://par.nsf.gov/biblio/10182302.

[8] A. Ruhela, S. Xu, K. V. Manian, H. Subramoni and D. K. Panda, "Analyzing and understanding the impact of interconnect performance on HPC, big data, and deep learning applications: A case study with infiniband EDR and HDR," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, New Orleans, United States, pp. 869–878, 2020.

[9] R. Beltman, S. Knossen, J. Hill and P. Grosso, "Using P4 and RDMA to collect telemetry data," in *IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*, Online conference, pp. 1–9, 2020.

[10] NVIDIA, "NVIDIA DeepOps," 2021. [Online]. Available: https://github.com/NVIDIA/deepops.

[11] M. Chadha, J. John and M. Gerndt, "Extending slurm for dynamic resource-aware adaptive batch scheduling," *arXiv preprint arXiv:2009.08289*, 2020.

[12] N. Zhou, Y. Georgiou, L. Zhong, H. Zhou and M. Pospieszny, "Container orchestration on HPC systems," in *IEEE Int. Conf. on Cloud Computing (CLOUD)*, Online conference, pp. 34–36, 2020.

[13] N. Mittal and S. Kumar, "Machine learning computation on multiple GPU's using CUDA and message passing interface," in *IEEE Int. Conf. on Power Energy, Environment and Intelligent Control (PEEIC)*, Greater Noida, India, pp. 18–22, 2019.

[14] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[15] J. Hines, "Stepping up to summit," *Computing in Science & Engineering*, vol. 20, no. 2, pp. 78–82, 2018.

[16] T. P. Morgan, "The clever machinations of Livermore's Sierra supercomputer," 2017. [online]. Available: https://www.nextplatform.com/2017/10/05/clever-machinations-livermores-sierra-supercomputer/.

[17] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song *et al.,* "The sunway taihulight supercomputer: System and applications," *Science China Information Sciences*, vol. 59, no. 7, pp. 1–16, 2016.

[18] X. Liao, L. Xiao, C. Yang, and Y. Lu, "Milkyway-2 supercomputer: System and application," *Frontiers of Computer Science*, vol. 28, no. 3, pp. 345–356, 2014.

[19] S. Belov, I. Kadochnikov, V. Korenkov, M. Matveev, D. Podgainy *et al.,* "High-performance computing platforms for organizing the educational process on the basis of the international school "data science," in *CEUR Workshop Proceedings*, vol. 2507, pp. 159, 2019.

[20] S. Nesmachnow and S. Iturriaga, "Cluster-UY: collaborative scientific high performance computing in Uruguay", in *Int. Conf. on Supercomputing in Mexico*, Monterrey, Mexico, pp. 188–202, 2019.
[21] M. Aldinucci, S. Bagnasco, S. Lusso, P. Pasteris, S. Rabellino *et al.,* "OCCAM: A flexible, multi-purpose and extendable HPC cluster," in *Journal of Physics: Conference Series*, vol. 898, pp. 082039, 2017.
[22] D. Pera, "Design and performance evaluation of a linux HPC cluster," *TASK QUARTERLY*, vol. 22, no. 2, pp. 113–123, 2018.
[23] A. Chazapis, J. -T. Acquaviva, A. Bilas, G. Gardikis, C. Kozanitis *et al.,* "EVOLVE: HPC and cloud enhanced testbed for extracting value from large-scale diverse data," in *Proc. of the ACM Int. Conf. on Computing Frontiers*, Online conference, pp. 200–205, 2021.
[24] J. Han, J. S. Shin, J. Kwon and J. Kim, "Cloud-native smartx intelligence cluster for ai-inspired hpc/hpda workloads," in *Proc. of the ACM/IEEE Supercomputing Conf.*, Denver, United States, 2019.
[25] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir and C. R. Das, "Kube-knots: Resource harvesting through dynamic container orchestration in GPU-based datacenters," in *Int. Conf. on Cluster Computing (CLUSTER)*, Albuquerque, United States, pp. 1–13, 2019.
[26] J. Shao, J. Ma, Y. Li, B. An and D. Cao, "GPU scheduling for short tasks in private cloud," in *Int. Conf. on Service-Oriented System Engineering (SOSE)*, San Francisco, United States, pp. 215–2155, 2019.
[27] Z. Chen, W. Quan, M. Wen, J. Fang, J. Yu *et al.,*"Deep learning research and development platform: Characterizing and scheduling with qos guarantees on gpu clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 34–50, 2019.
[28] F. Filippini, M. Lattuada, A. Jahani, M. Ciavotta, D. Ardagna *et al.,* "Hierarchical scheduling in on-demand gpu-as-a-service systems," in *Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Online conference, pp. 125–132, 2020.
[29] T. Habuza, K. Khalil, N. Zaki, F. Alnajjar and M. Gochoo, "Web-based multi-user concurrent job scheduling system on the shared computing resource objects," in *Int. Conf. on Innovations in Information Technology (IIT)*, Online conference, pp. 221–226, 2020.
[30] Y. Luan, X. Chen, H. Zhao, Z. Yang, and Y. Dai, "Sched2: Scheduling deep learning training via deep reinforcement learning," in *Global Communications Conf. (GLOBECOM)*, Waikoloa, United States, pp. 1–7, 2019.
[31] S. Wang, O. J. Gonzalez, X. Zhou, T. Williams, B. D. Friedman *et al.,* "An efficient and non-intrusive GPU scheduling framework for deep learning training systems," in *SC20: Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Online conference, pp. 1–13, 2020.
[32] M. Abdullah, W. Iqbal, F. Bukhari and A. Erradi, "Diminishing returns and deep learning for adaptive CPU resource allocation of containers," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2052–2063, 2020.
[33] H. Zhao, W. Cui, Q. Chen, J. Leng, K. Yu *et al.,* "CODA: Improving resource utilization by slimming and co-locating DNN and CPU jobs," in *Int. Conf. on Distributed Computing Systems (ICDCS)*, Singapore, Singapore, pp. 853–863, 2020.
[34] J. Lin, D. Cui, Z. Peng, Q. Li and J. He, "A two-stage framework for the multi-user multi-data center job scheduling and resource allocation," *IEEE Access*, vol. 8, pp. 197863–197874, 2020.
[35] K. Zhang, Z. Yang and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *arXiv preprint arXiv:1911.10635*, 2019.