Job placement using reinforcement learning in GPU virtualization environment

Jisun Oh & Yoonhee Kim

Cluster Computing
The Journal of Networks, Software Tools and Applications

ISSN 1386-7857

Cluster Comput
DOI 10.1007/s10586-019-03044-7

Springer

Cluster Computing
https://doi.org/10.1007/s10586-019-03044-7

Job placement using reinforcement learning in GPU virtualization environment

Jisun Oh[1] · Yoonhee Kim[1]

Received: 25 December 2019 / Revised: 25 December 2019 / Accepted: 31 December 2019

Abstract
Graphics Processing Units (GPU) are widely used for high-speed processes in the computational science areas of biology, chemistry, meteorology, etc. and the machine learning areas of image and video analysis. Recently, data centers and cloud companies have adopted GPUs to provide them as computing resources. Because the majority of cloud providers allocate the GPU resource to users in an exclusive access method, the allocated GPU resource may not be all used. Although the method of allocating a GPU resource to multiple users for sharing can increase the resource utilization, performance degradation may occur in individual jobs because of interference between different jobs. It is difficult for a cloud provider to predict or control the performance of various applications executed on various cloud resources by considering their characteristics heuristically. Therefore, an intelligent job placement technique is required to minimize the interference between different jobs and increase resource utilization. This study defines the resource utilization history of applications and proposes a reinforcement learning-based job placement technique, which uses it as an input. For resource utilization history learning, a deep reinforcement learning model (DQN) is used. As a result of learning, the current resource's state is not exceeded, and the resource is still provided by predicting which commonly placed jobs will have less impact on the total performance when executed simultaneously. This approach prevents the performance degradation of applications with diverse execution characteristics and increases the resource utilization by executing the applications while sharing the resources. The superiority of this study is demonstrated by using the proposed learning method and other methods to analyze workloads with various resource utilization characteristics. Through the experiments, it is proven that the proposed method facilitates a reduction of the total execution time and the effective use of resources, while the maintaining performance.

Keywords GPU · DQN learning · Interference prediction · Multiple job placement

1 Introduction

A Graphics Processing Unit (GPU) consists of thousands of processing cores and performs parallel procession operations at a high level. Based on the advantage of highly accelerated processing of computing tasks, general-purpose GPUs (GPGPUs) are broadly used in diverse areas such as machine learning (ML) and high-performance computing (HPC) applications. Hence, cloud and server infrastructure providers provide GPU servers to users for the execution of various applications. Many large cloud providers, such as Amazon EC2 [1], Nimbix [2], Microsoft Azure [3], and Alibaba [4] provide GPU services. When a user requests a GPU, these service providers (e.g., Kubernetes [5], Mesos [6]) grant exclusive access rights to the user instead of on-demand access. Such an exclusive access right can reduce the total system throughput as it blocks access to the whole GPU resource for other jobs, when the waiting time for an application is already long [7]. As a result, the GPU may not be used for sufficient amounts of time, and the provider ends up with an idle computing resource.

The objectives of computing infrastructure providers are the attainment of a high resource usage and a reduction in

✉ Yoonhee Kim
yulan@sookmyung.ac.kr

Jisun Oh
jsoh8088@gmail.com

[1] Sookmyung Women's University, Seoul, South Korea

Published online: 09 January 2020

operating costs. To accomplish these objectives, the majority of existing the computing infrastructures are facilitating the efficient sharing of resources such as CPU and memory [8]. Among various resources provided in physical/virtual cluster systems, the process of using the GPU resource increases the cost and energy use [9]. Therefore, the GPU resources should also be shared for the economic benefits of the operation. The operating cost can be reduced by securing a minimum number of GPU instances and sharing the GPU across multiple systems [10, 11]. Furthermore, GPU prices are expensive, but GPU resources can be provided less expensively if multiple users share it. In addition, GPU resource utilization is increased whenever many applications are executed together by sharing a GPU. Recently, NVIDIA has provided a Multi-Process Service (MPS) [12] that executes multiple applications in one process by GPU sharing. However, this technology can be helpful for performance improvement if, and only if the kernel pattern of the application is known. There are studies on the scheduling of co-execution tasks of applications in GPU resources: the co-placement method of applications is based on monitoring [13–15], and a weight-based placement method using the GPU use profiling information of applications [16].

However, although resource utilization can be increased by placing several applications together on a server, performance degradation may occur in the total performance due to the competition which occurs for the shared resources when several jobs are executed simultaneously. Furthermore, performance prediction becomes more difficult as interferences can occur between different applications to ensure they are each receiving sufficient resources. This is because all applications share and use basic resources such as cache, streaming multiprocessor (SM), and I/O as well as the resources (CPU, GPU, and memory) usually considered by the scheduler [17, 18]. Nevertheless, in reality, various characteristics of resource use exist as well as the complex environment and interactions. Therefore, it is difficult to predict the interference and the performance just by the resource usage of a job.

This paper proposes a reinforcement learning-applied data placement method based on the job history of HPC and ML applications in the GPU cluster environment. The job history is defined according to the resource, for which most of the resource competition occurs, and using the history as input, reinforcement learning-applied data placement is performed to reduce the execution time and increase the resource utilization. Furthermore, this paper analyzes the execution time and training overhead by job when co-placement is performed, and as a result, each job's performance and the total performance show less degradation. The major research contents of this paper are as follows.

- The execution characteristics of various applications are described, and the relationship between the performance and the resource usage is shown, with interference occurring when the applications are executed together.
- The resource utilization history of the application to be used for learning is defined (GPU application prior information, GPU application profiling information, and cluster environment information).
- A reinforcement learning-based job placement method is proposed, which uses the defined resource utilization history as input.
- The performance of the proposed method is evaluated through experiments by comparing it with the performance of other methods.

This paper is organized as follows. Section 2 shows the characteristics of the applications and their relationships with the performance when the resource is shared. Section 3 describes the reinforcement learning-based job placement method which uses job history. Section 4 describes experiments performed based on the proposed data placement method and the analyzes of the results. Section 5 summarizes the studies on the resource sharing method and the reinforcement learning-applied data placement method, and comparatively analyzes them with the study described in this paper. Lastly, Section 6 provides the conclusion.

2 Background and motivation

This section describes the resource utilization characteristics of HPC and ML applications and explains the relationship between the sharing of the resource and the performance degradation occurring when applications are executed together in a GPU cluster environment. Through this, the need for reinforcement learning for job placement is shown, especially regarding the performance prediction.

2.1 GPGPU applications' characteristics

As GPUs have been widely used in the HPC and ML areas, their functions have expanded to those of GPGPU. These applications of GPUs show various resource utilization characteristics in the GPU cluster environment. In the case of applications in the two areas, GPU is used for massive data processing, but they have different characteristics from each other. In the case of ML applications, the execution time varies depending on the size of the data set, the number of layers, and batch size, but the GPU resource is used consistently during execution [19]. Figure 1 shows the GPU and GPU memory utilization patterns when the

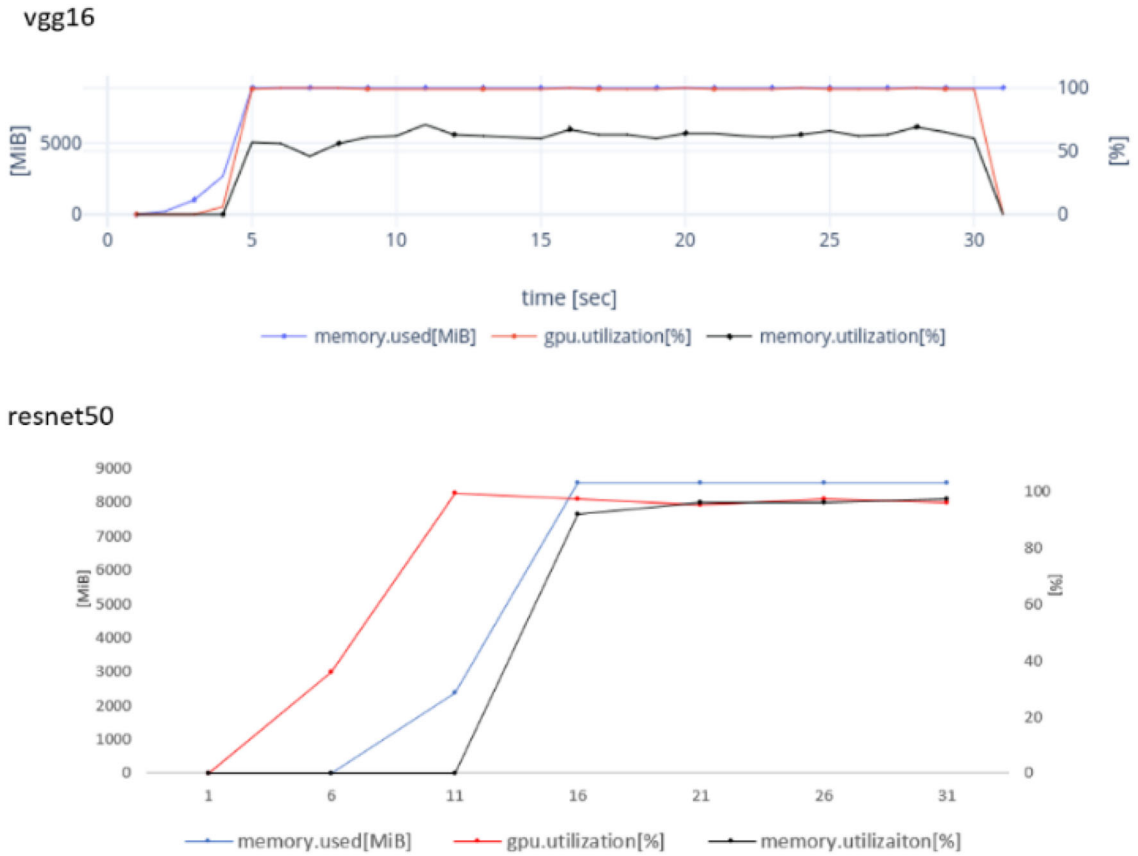


Fig. 1 Examples of resource utilization patterns for the ML applications of vgg16 and resnet50

ML-applied vgg16 and resnet50 of the TensorFlow benchmark [20] are executed. Except for during the loading of the data set onto the GPU memory at the beginning of execution, the GPU and GPU memory utilization are almost constant during the learning and the patterns are maintained until termination is reached.

In the case of HPC applications, there is a subordinate process that has to be executed for scientific application analysis. HPC applications have various GPU resource utilization patterns because they are executed according to different preprocessing processes and the analysis process of each application differs as well. Figure 2 shows the resource utilization patterns of LAMMPS [21] and QMCPACK [22] which are among the HPC applications. For the LAMMPS application, the average GPU utilization is about 40% during execution. The GPU memory utilization increases gradually to about 363 MB - 8,499 MB, and 100% of the GPU is used when the GPU memory utilization is at its peak. In the case of the QMCPACK application, GPU utilization increases in steps of three stages. The GPU memory is not used until the middle of the application execution, and after the middle (about 95 s), it shows a sharp increase to about 5,135 MB.

The ML applications have consistent resource utilization patterns during learning and analysis, and the HPC applications have various resource utilization patterns depending on the execution stage. Therefore, the next section analyzes the relationship between the resource and the performance when multiple applications are co-executed.

2.2 Performance vs resource usage with interference

When multiple applications are executed and using the resources in the GPU cluster environment, it is difficult to predict the performance due to the different resource utilization patterns of the applications. Furthermore, even if all applications are provided with sufficient resources, they can interfere with each other [23]. This is because, other resources, like core and I/O bandwidth are shared and used in addition to the resources usually considered by the scheduler (CPU, GPU, and memory) [7].

In this study several applications are placed together for jobs and compared according to the performance by considering the shared resources during execution. The comparison results show that the performance may be affected by a sharing of resources, and that these results are not

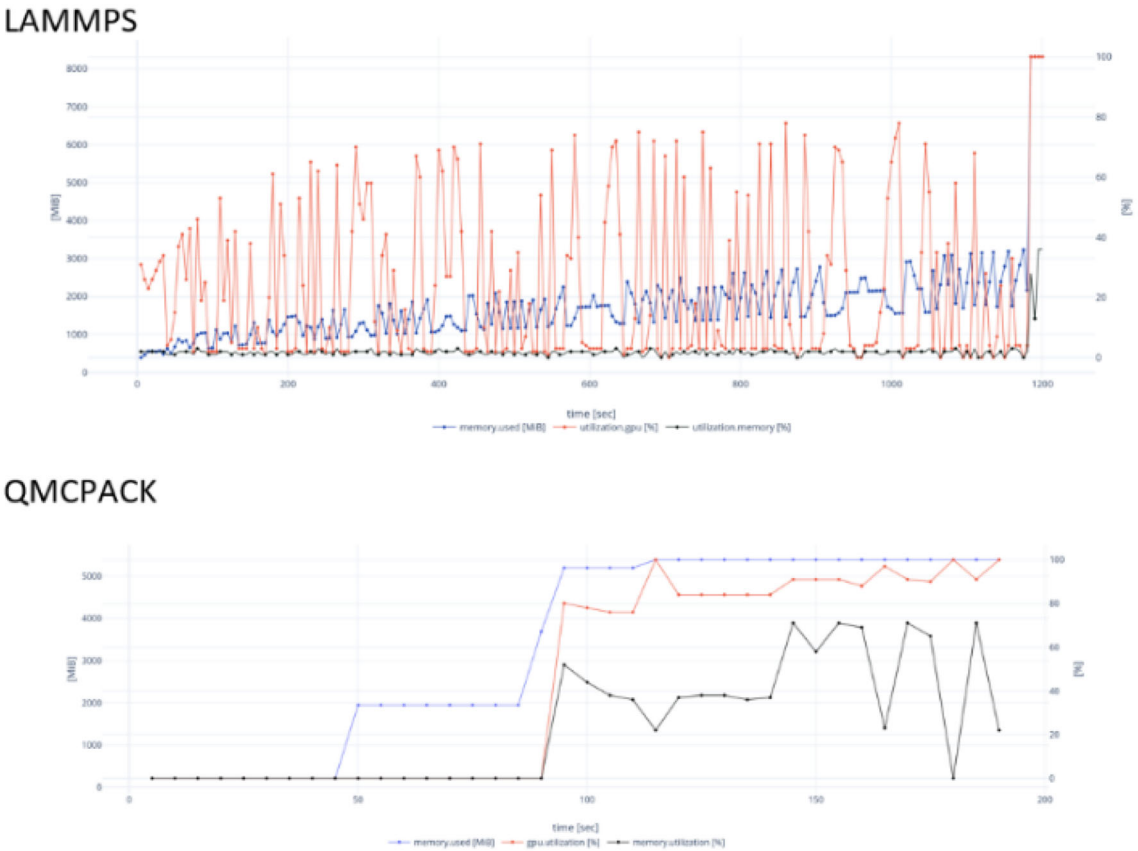


Fig. 2 Examples of a resource utilization pattern for HPC applications of LAMMPS and QMCPACK

sufficient for accurate performance prediction. For the comparison, we used three deep-learning applications (CNN, vgg16, googlenet) of TensorFlow benchmark showing similar resource utilization characteristics and two HPC applications (LAMMPS and GROMACS) of NGC (Nvidia GPU Cloud) [24].

Fiure 3 shows the execution time and GPU memory, core, and I/O usages when two applications were placed and executed together. For example, as the red color becomes darker in the diagram of the execution time, the execution increases. In contrast, the darker the green is, the higher the usage in the diagram of the GPU memory usage level, which is drawn on a scale of 0 to 5. In the case of applications (LAMMPS, LAMMPS), (vgg16, CNN), (CNN, vgg16), and (vgg16, vgg16), memory shortage occurs and consequently, affects the execution time. However, except for these cases, it is difficult to ascertain the co-execution performance of applications through the usage of GPU memory, core, and I/O. For example, in the case of (googlenet, vgg16), the performance degradation level of the execution time is 4, but for the usage level of GPU memory, it is 3, while that of the GPU core and I/O are 2 and 1, respectively. This indicates that when executing several applications simultaneously, the shared use of each resource leads to complex interactions and consequently, they affect the performance in complicated ways. Scheduling that predicts the performance by using a conventional heuristic method through the usages of each resource is an NP-hard problem. Therefore, the performance should be predicted by considering the characteristics of all applications and the status of the resource.

3 DqnGPU design

This section introduces the reinforcement learning-based job placement method for multiple jobs of GPU applications in the GPU cluster environment. First, detailed job history items are defined for each resource the application used, for learning. The reinforcement learning-based job history learning model is described, and the resource sharing job placement service model is explained.

3.1 Resource utilization history items of GPU application

When applications are placed and executed together, not only does performance degradation occur, but it is also

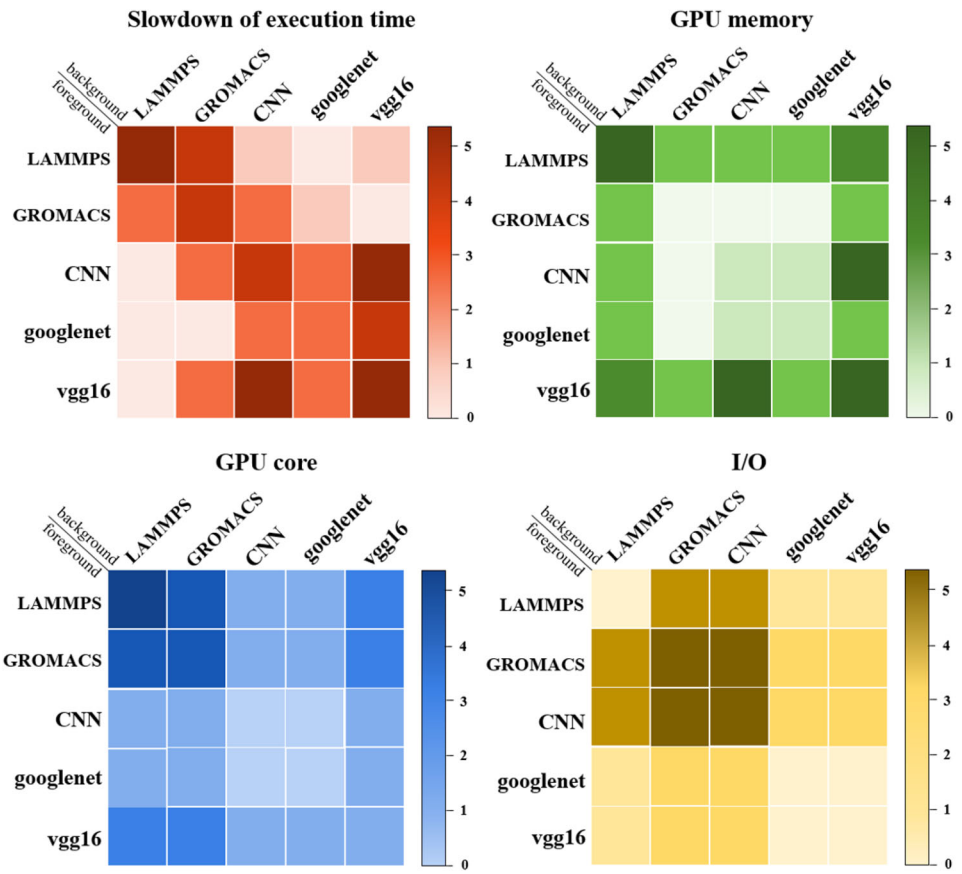


Fig. 3 Comparison of the relationship between the performance and the resource usage when applications are placed and executed together

difficult to manage the execution according to the complex computing system environment. Reinforcement learning, which observes the environment and determines an optimal policy based on the observation results, is suitable for such a cloud and cluster computing system [25, 26].

The resource utilization history items of applications to be collected for reinforcement learning were defined. Table 1 shows the items of the job history for GPU applications in the GPU cluster environment, and they are classified largely into three categories. First, prior information of GPU applications is the information that a user submits to the system to execute the application, and it requires the application name, application type, input file size, and input parameters. Second, the profiling information of the GPU application shows the information collected online when an application is executed. It includes the resources used during the execution of the application and the execution time. The collected resources are the items that can cause performance degradation when multiple GPU applications are executed together, because then resource competition occurs. The profiling information of the resources includes GPU, GPU memory, GPU core, and PCIe usages. The usage of each item is collected by monitoring for a certain time duration periodically. Third, the cluster environment information is the information of each resource used for the execution of jobs. The GPU application profiling information and the environment information collected every hour are used to prevent

Table 1 The resource utilization history items of GPU applications

Category	Attribute
GPU application prior information	Application name
	Application type
	Input file size
	Input parameter
GPU application profiling information	GPU utilization
	GPU memory usage
	GPU core usage
	PCIe throughput
	Execution time
Cluster environment information	Node name
	GPU card
	GPU architecture
	GPU memory
	GPU core size
	PCIe bandwidth

excessive placement on the resources when multiple jobs are placed.

3.2 DqnGPU model

The goal of reinforcement learning is to learn a strategy that takes better decisions through direct interaction with the system [27]. Reinforcement learning has the advantage that a strategy can be applied without prior knowledge because it learns repetitively which action is performed in a certain situation. However, the conventional reinforcement learning techniques have limitations in storing numerous input data in a table format and solving problems. DQN is a technique introduced first by the DeepMind team which developed AlphaGo [28], and uses Deep Neural Network (DNN) as a function approximator to resolve the problem of reinforcement learning.

The learning model of this paper was constructed by modifying a conventional DQN model-based job placement model deepRM [29]. The conventional deepRM (1) generates a virtual job, and (2) the virtual job targeting the CPU and memory resources has certain resource usages only. Furthermore, (3) only the execution of a single job placement is considered in the learning result. Therefore, this study proposes a multiple job placement technique through a DQN model that reflects various resource utilization characteristics targeting real application jobs. (1) Targeting real GPU HPC and ML applications, (2) the applications' changing resource utilization histories of GPU, GPU memory, GPU core, and PCIe are learned. (3) The multiple job placement technique is considered by generating virtual job slots of real applications. (4) The policy parameters are modified for a smaller distribution deviation.

3.2.1 Action space

Figure 4 shows the overall DQN structure, in which the agent interacts with the environment and repeats learning, rewarding, and observing. At step t, the agent observes the current s_t and receives a request to perform an action a. After performing the action, the state transitions to s_{t+1} and the agent receives the reward r_t. Here, because the state transition and the reward are affected only by the action that the agent performed, they have the Markov property. The goal of an agent is to maximize the cumulative discounted reward while repeating the learning process. The cumulative discounted reward is expressed as $J(\theta) = max\mathrm{E}[\sum_{t=0}^{\infty} \Gamma^t r_t]$ and the discount factor γ has a value between 0 and 1. Here, the action is selected based on the agent's policy, and for the policy, π: $\pi(s,a) \rightarrow [0,1]$; $\pi(s,a)$ is expressed as a propability of action a being executed in the state s. However, as mentioned earlier, it is difficult to store numerous state,action values in a table. Therefore, a function approximator DNN is used to obtain the policy parameter θ. Accordingly, the policy can be expressed as $\pi_\theta(s,a)$.

$$\nabla_\theta J(\theta) = \mathrm{E}_{\pi\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta log\pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right] \quad (1)$$

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta log\pi_\theta(s_t,a_t)(Q^{\pi_\theta}(s_t,a_t) - V^{\pi_\theta}(s_t)) \quad (2)$$

Therefore, the final goal of DQN is to maximize the expected cumulative discounted reward $Q^{\pi_\theta}(s,a)$ when action a is performed in the state s according to the policy π_θ (Eq. 1). Here, in the reinforcement learning, the policy parameter θ is updated by using the gradient descent [37] to reduce the deviation of the Q value distribution. The conventional gradient descent method uses only the empirical cumulative discounted reward V obtained according to the policy instead of Q which can be biased toward an incorrect value. This study predicts the gradient descent value by subtracting the empirical cumulative discounted reward V^{π_θ} obtained according to the policy from the predicted cumulative discounted reward Q^{π_θ} for the policy parameter having lower distribution (Eq. 2).

Algorithm 1 shows the overall DQN model training algorithm with the modified parameters.

Algorithm 1 DqnGPU Model Training

```
Initialize Q(s,a) and Model(s,a), ∀a ∈ S, ∀s ∈ A(s)
for each iteration do
    for each iteration do
        for running job's episode i = 1, ..., N do
            Select a_i = max_a Q(s_i, a)
            Execute action a_i in emulator and observe reward r_i and image x_{i+1}
            Set s_{i+1} = s_i, a_i, x_{i+1} and update reward r_i, policy q_i
        end for
        Compute returns: v_t^i = Σ_{s=t}^{Li} γ^{s-t} r_s^i
        for t = i to L do
            for i = 1 to N do
                △θ ← △θ + α∇_θ log π_θ(s_t^i, a_t^i)(q_t^i − v_t^i)
            end for
        end for
    end for
    θ ← θ + △θ
end for
```

3.2.2 State space

The application's resource utilization history defined in Table 1 of Sect. 3.1 is used as input for the DQN model learning for the co-placement of jobs. As shown in Fig. 4, the model's input state $s = (j, R)$ includes the job vector j and the cluster's resource vector R. Through the cluster's

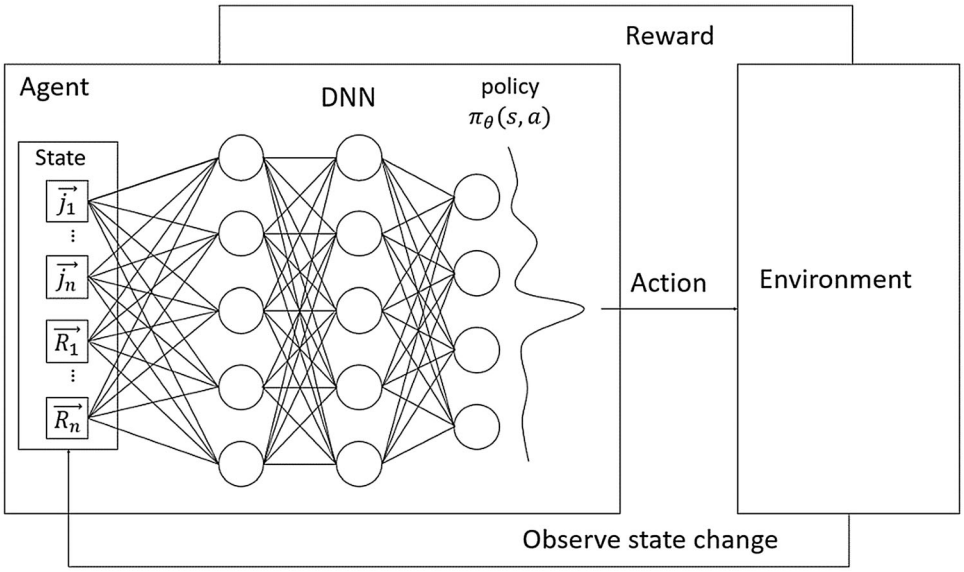


Fig. 4 Proposed DNN-based DQN structure

resource environment history items, a cluster resource can be shown using a four-tuple set, as shown in Eq. 3, and it indicates the usable resource amount of each resource of the cluster. The job j is the profiling history information of the resource collected at every timestep T for each job. j_i has the information of the resource usage changing at every timestep T_j of the corresponding application APP and can be expressed by Eqs. 4 and 5.

$$R \in \{R_{gpu}, R_{gpumem}, R_{gpucore}, R_{pcie}\} \tag{3}$$

$$j_i \in \{APP_{id}, T_j, r_{j.gpu}, \mathbf{r_j}\} \tag{4}$$

$$\mathbf{r_j} = (r_{j.gpu}, r_{j.gpumem}, r_{j.gpucore}, r_{j.pcie}) \tag{5}$$

In the state space, the cluster environment's currently allocated resources and the resource profiling history of the job in a reserved and waiting state are shown in separate images. Figure 6 shows examples of the system's state space. A cluster's image consists of four images for each resource, and shows the allocation of each resource for the reserved jobs at an interval of time T. A job image shows the resource usage at an interval of step T based on the application's profiling history collected online. For example, the purple-colored job in Fig. 6 shows the resource requirements for the following five time intervals, respectively: 0, (1, 1, 0, 1), 1, (2, 3, 2, 1), 2, (3, 0, 2, 1), 3, (0, 0, 3, 1), 4, (0, 0, 0, 1). Based on this, an image for each resource of a job is constructed as a table composed of 0 s and 1 s. When there is a resource used at step T, a job's image is expressed using a table filled with 1 s for the amount of resources used and 0s for the remaining amount. As such, cluster images are constructed based on the images of multiple jobs to be placed together, and here, the following conditions should be satisfied:

$$\sum_j r_{j.gpu} \cdot T_j \le R_{gpu}, \sum_j r_{j.gpumem} \cdot T_j \le R_{gpumem},$$
$$\times \sum_j r_{j.gpucore} \cdot T_j \le R_{gpucore}, \sum_j r_{j.pcie} \cdot T_j \le R_{pcie}.$$

Therefore, as shown in an example in Fig. 5, a cluster

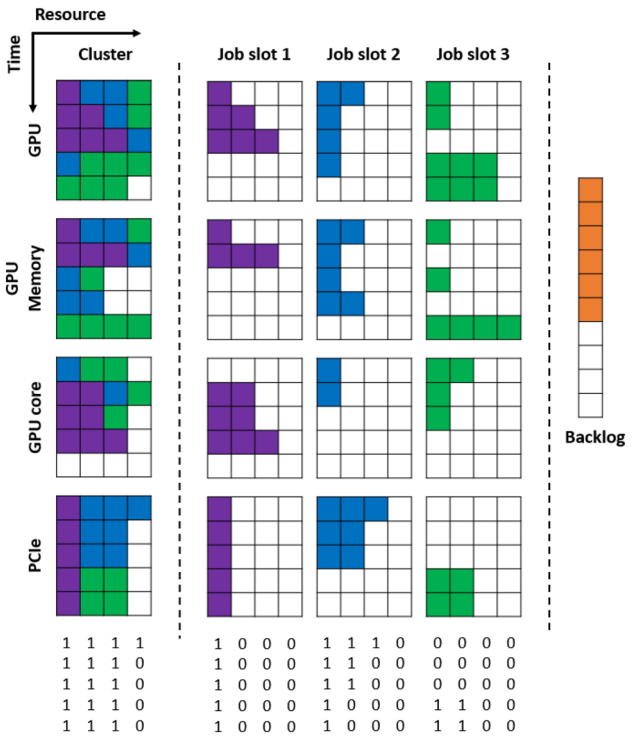


Fig. 5 Examples of state space for the proposed method's four resources and waiting job slots

image and three job image tables can be expressed for the PCIe resource.

Because it is desirable to express the inputs of a neural network in the fixed state, as many inputs as the number of job image slots M are used. As shown in Fig. 5, if three job image slots exist, three jobs can be used as inputs for learning. The result obtained in learning is used to determine a next action a. The action space is expressed as $a = 0, 1, \ldots, M$. When $a = 1$, it means that the job allocated to the first job slot is to be executed. When $a = 0$, it indicates that the action is pending and there is no available space in the cluster image for the job image. Therefore, while proceeding in an interval of step T, an appropriate job a is decided and executed, and as the resource's cluster image is updated, the process is repeated. The information regarding the job after M is included in the backlog components of the state space. If an empty job slot exists, it is filled with the job taken from the backlog. In the above method, however, only one job can be executed according to the result a. Therefore, jobs cannot be placed together and executed.

In our study, an input space of the neural network was constructed for multiple job placements. Virtual job image slots are generated according to the number of job image slots M. For example, suppose three job image slots exist in the state space and the jobs of a, b, and c exist in each slot. Then, a virtual job image slot is composed of a, b, c, ab, ac, bc, and abc, seven in total, and a table is generated according to the resource usage of the composed jobs. The learning is performed using the virtual job image slot as input. As a result of learning, jobs a and b are placed together and executed in the case of an action a= ab. Therefore, for the input into the neural network learning for co-placement, a virtual job image slot is constructed with $\sum_{r=1}^{M} {}_M C_x = 2^M - 1$, the combination of M job image slots, is used. As a result of learning, one case among the M combinations of applications is generated, and based on this, the jobs to be co-placed are determined and executed.

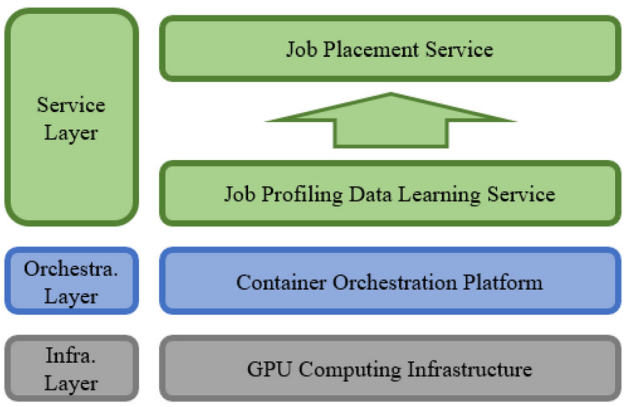


Fig. 6 The learning-based job placement service model

3.3 DqnGPU model-based job placement method

Figure 6 shows the structural diagram of the proposed learning-based job placement service model. This paper targeted the GPU computing container for the infrastructure and used kubernetes for the orchestration platform. The proposed service model provides the job placement service that reflects the job history learning model proposed in Sect. 3.2 and the learning results.

The job placement service places and executes the jobs to be co-placed, i.e., the action values obtained according to the system environment and the characteristics of the applications inferred earlier; and monitors the status of the system and jobs. Based on this, a job's failure, termination, etc. are identified and the next job is provided a place on the resource as soon as it is available.

This study assumes that profiling information exists for the GPU applications a user wants to execute. Furthermore, it is assumed that the container image containing the compiler and library of the application to be executed exist in the registry, which is the image storage.

Algorithm 2 Job Placement Algorithm

1: Submit Application APP_i
2: Find $ProfilingData_i$, $ClusterStatus$
3: Set $InputData_i = \{ProfilingData_i, ClusterStatus\}$
4: Order of $jobs_j$ and $|jobs_j| \leftarrow DqnGPU(InputData_i)$
5: **for each** $jobs_j$ to $jobs_{j\ldots n}$ **do**
6: Assign $jobs_i$ to GPU
7: MonitoringSystem()
8: **end for**

The algorithm in Algorithm 2 shows the overall flow of placing jobs that will be co-executed, while reflecting the status of the cluster resource and profiling information history which the user has submitted based on the reinforcement learning model. The user submits an application that he/she wants to execute (line 1). The submitted application's profiling data (*ProfilingData*) and the system's status data (*ClusterStatus*) are checked (line 2) and the input data that will be used as input for the learning model are set up (line 3).

$$ProfilingData = \text{Eq.}(4)'s\ j_i \in \{App_i d, T_j, \mathbf{r_j}\}$$

$$ClusterStatus = \text{Eq.}(3)'s\ R \in \{R_{gpu}, R_{gpumem}, R_{gpucore}, R_{pcie}\}$$

Afterwards, by applying the reinforcement learning, this algorithm decides the order of the jobs of the entire workload queue and the jobs that will be co-placed (line 4). The jobs are co-placed and executed according to the inferenced order of jobs (line 6), and the status of the jobs

is monitored (line 7). This is repeated until all the jobs are placed (line 5–8).

4 Experiments

The experiment was conducted and the results were analyzed to evaluate the performance of the proposed method. In this section, the experiment's target applications and the experimental environment are described, and the comparison method is introduced. Lastly, the performed experiment is described, and the results are analyzed.

4.1 Target applications and experimental environment

The following HPC applications and ML benchmark applications were used to verify the performance of the proposed job placement method. For HPC applications, four applications provided by NGC and seven TensorFlow benchmark applications are used as the target applications of the experiment. The names of the applications used in the experiment are shown in Table 2.

Five workloads were generated for each workload according to the resource utilization history of the applications mentioned above, for the experiment. A total of five workloads were configured: GPU_heavy and GPU-light workloads based on an average GPU utilization level of 40%; GPU memory_heavy and GPU memory_light workloads based on an average GPU memory level of 50%; and a random workload, in which all applications were randomly mixed. For each workload, a total of 30 applications were generated randomly based on the configuration criteria of the workload. Then, the total execution time, average GPU utilization level, and average GPU memory utilization level were measured.

For the experiment, a GPU-based container cluster environment was constructed, and for the cluster environment, Kubernetes, a container orchestration platform was used. Kubernetes is a Container as a Service (CasS) type of cluster computing open source project, and provides a service to automate the container's placement, scaling, and operation. Large companies such as Amazon, Alibaba, Samsung SDS, baidu, Huawei, and IBM have adopted and are using it for their cloud operations.

Table 2 The target applications in the experiment

HPC	ML	
LAMMPS	googlenet	resnet50
GROMACS [30]	alexnet	resnet101
QMCPACK	vgg16	resnet152
HOOMD [31]	vgg11	

Kubernetes' basic device plugin k8s was modified for multiple job placements of a node. When the conventional plugin is used, servers in the Kubernetes environment, i.e., multiple servers for the execution of multiple jobs are not created. Furthermore, the co-execution of multiple jobs in a single card of the node is not supported. Therefore, the plugin was modified to facilitate the creation of multiple servers and the execution of multiple jobs in the experiment of this study. It was modified by using an open-source Alibaba fake GPU [32] as a reference. Furthermore, the jobs were executed by using a NVIDIA docker container [33]. The specifications of the computing node used in the experiment are shown in Table 3 below.

4.2 Experiments and analysis of the results

The performance of each workload is analyzed, and each job's speed degradation and training overhead are analyzed comparatively in order to analyze the performance of the proposed job placement method. For comparison with the proposed method of this study, four job placement methods are used: SJF, which is a single job placement method; Tetris [34], which is a multiple job placement method; and DeepRM Max and DeepRM Mean, which are placement methods applying the reinforcement learning. The descriptions of each job placement method are as follows:

- SJF: jobs are placed in increasing order of job execution time
- Tetris: jobs are placed in a packing method according to the applications' job usage and resource availability
- DqnGPU: the proposed placement method of this study
- DeepRM Max: jobs are placed by applying the reinforcement learning, in which the maximum resource usage value of a job is used as an input
- DeepRM Mean: jobs are placed by applying the reinforcement learning, in which the average resource usage value of a job is used as an input

4.2.1 Execution result analysis by workload

In the experiment, the results of the execution using the results of job placement inferenced through the proposed learning model are compared with the results of the other placement methods.

A GPU_heavy workload consists of vgg11, vgg16, resnet50, resnet101, resnet152, qmcpack, hoomd, and lammps applications that show an average GPU utilization of 40% or higher. Figure 7a shows the execution results of a GPU_heavy workload. When jobs are placed using the SJF method, the execution time is 4807 s, the shortest.

Table 3 The cluster environment information

	Node details	
	CPU (master)	GPU (node)
Architecture	Intel Core™ i7-5820K	Nvidia GeForce Titan Xp D5x
Core clock	3.30 GHz	1.58 GHz
Num of cores	6	3840
Mem. size	32 GB	12 GB
Threading API	–	NVIDIA CUDA 10.0
PCIe bandwidth	–	32 GB/s
OS	Ubuntu 16.04.6 LTS	Ubuntu 16.04.6 LTS

However, the average GPU utilization is 70.50% and the average GPU memory utilization is 36.44%, which are the lowest values of this study. Among the methods of executing multiple jobs, the method proposed in this study shows the highest performance with 4891 s, and the average GPU utilization and the average GPU memory utilization show high resource utilizations with 82.09% and 59.13%, respectively. In the case of DeepRM Mean, the longest execution time is shown with 5530 s, which is due to a memory problem (OOM: out of memory) occurring when the GPU memory is used fully, and the failed application is executed again.

A GPU_light workload consists of alexnet and gromacs applications that show an average GPU utilization of less than 40%. Figure 7b shows the execution results of a GPU_light workload. For executing applications with a low GPU utilization, the placement of the SJF method shows the lowest performance (2295 s) and resource utilizations. It seems because each application is placed alone, the SJF method increases the execution time compared to the other methods which are capable of co-execution. For the application, gromacs, the execution time is longer than that of alexnet, but because it uses less GPU memory, up to three jobs can be executed simultaneously when it is placed. The placement method proposed in this study shows the highest performance with an execution time of 1214 s, GPU utilization of 89.31% and GPU memory utilization of 88.42%, and based on the placement using the reinforcement learning, the GPU memory can be used up to a maximum of 11,755 MB.

GPU memory_heavy workload consists of vgg11, vgg16, resnet50, resnet101, and resnet152 applications that show the average GPU memory utilization of 50% or higher. Figure 7c shows the execution results of GPU memory_heavy workloads. For SJF, the execution time is the shortest, followed by the execution time of DqnGPU. It seems that the execution time has increased because when the executed applications are co-placed, resource competition is induced due to their high GPU utilizations. However, when compared to the other co-execution placement methods, it has the shortest execution time with 1214 s and the highest resource utilizations with 89.31% and 88.42%, respectively.

GPU memory_light workloads consists of alexnet, googlenet, qmcpack, hoomd, gromacs applications that show the average GPU memory utilization of less than 50%. Figure 7d exhibits the execution results of the GPU memory_light workload, and the method proposed in this study shows the highest performance in the aspect of execution time. It has the shortest execution time with 3928 s. This is because gromacs and qmcpack applications, which have low GPU memory utilization but a long execution time, have been placed and executed together based on the learning results.

A random workload consists of all applications used in the experiments. Figure 7e shows the execution results of a random workload. When the method proposed in this study is used, the execution time is 2481 s, the GPU utilization is 62.16%, and the GPU memory utilization is 20.45%. Regarding the execution time aspect, it shows better performance than the other co-execution placement methods. However, the GPU utilization is lower than that of Tetris and DeepRM Max when compared with the other workload experiments, and the GPU memory utilization is similar to that of SJF. From these results, it can be seen that when multiple applications exist, the impact of resource competition may be reflected in a variety of ways if they are co-executed because various resource utilization characteristics exist.

4.2.2 Speed degradation analysis of jobs

In order to evaluate the performance by using the random workload used in the above experiment, the speed degradation was analyzed for a total of 30 jobs with respect to the four co-execution placement methods. The evaluation was performed by classifying them into multiple categories—no speed degradation (2% error), less than 10% speed degradation, greater than or equal to 10% and less than 20% degradation, and greater than or equal to 20%

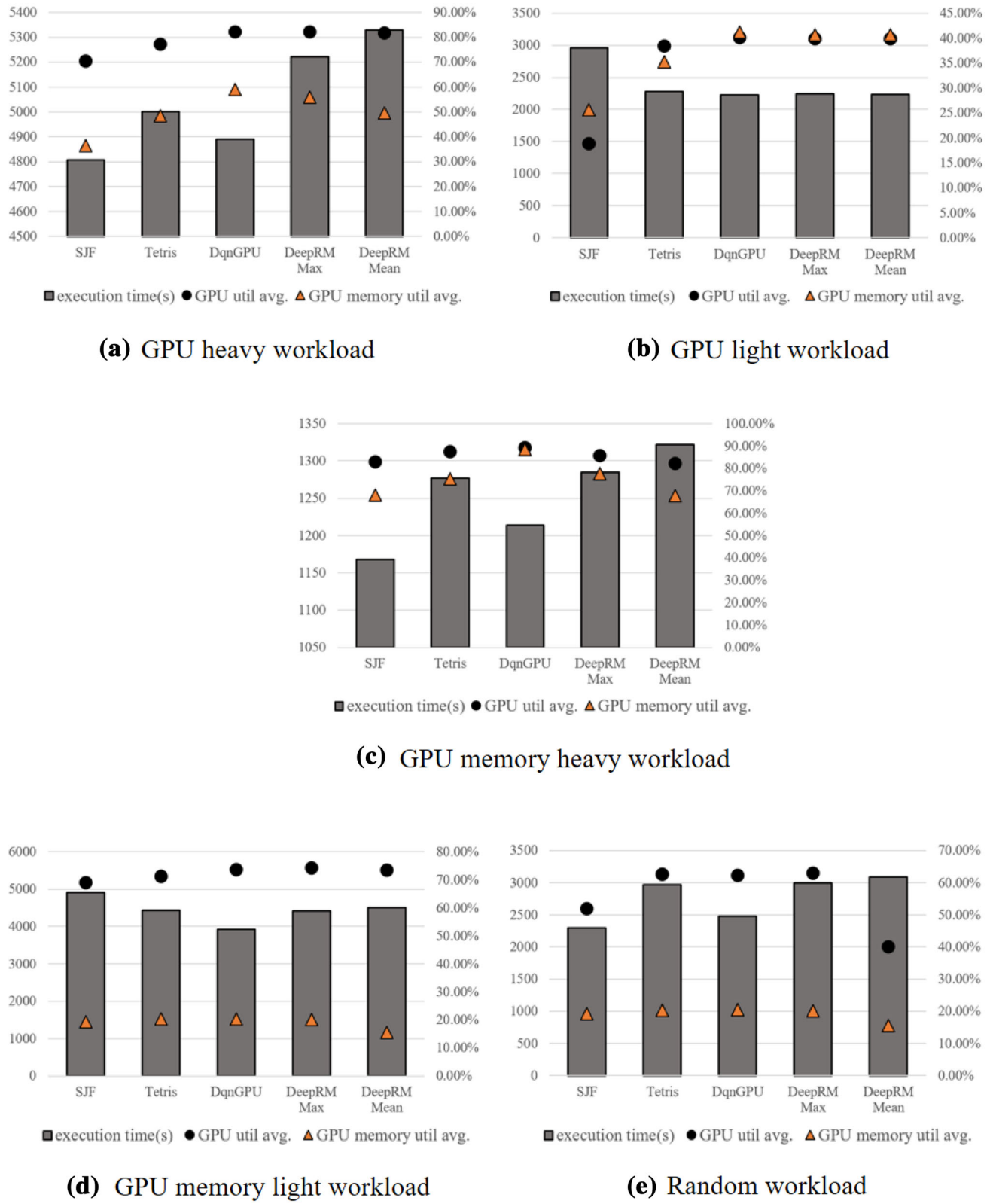


Fig. 7 Comparison of execution time, GPU utilization, GPU memory utilization between job placement methods by workload

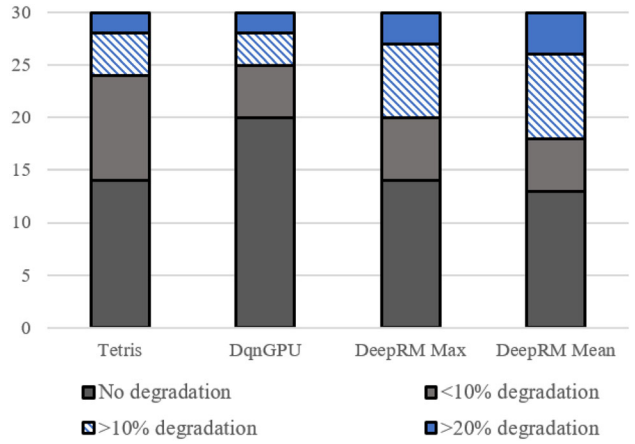


Fig. 8 The speed degradation comparison of 30 jobs using the four co-placement methods

degradation based on the execution time of a single executed application.

Figure 8 shows the speed degradation of each job for the four methods for the execution of the random workload. The method proposed in this study showed that no speed degradation occurred in a total of 20 jobs. Because the learning was performed according to the resource utilization, which changed with execution time, and the jobs were placed in a range not exceeding the actual limit of resources, the speed degradation caused by the resource competition was small. However, the occurring speed degradation implies that the characteristics, various conditions, or the environment of the executed applications, have an influence in addition to those of the resources, which was not considered before the co-execution. In the cases of Tetris and DeepRM Max methods, no speed degradation occurred for 14 jobs. In the case of DeepRM Mean, 12 jobs showed a speed degradation of 10% or larger, corresponding to the largest number of jobs. In the case of DeepRM Mean, because jobs are placed according to the mean resource values, many jobs showed speed degradations because of OOM problems occurring during co-execution.

4.2.3 Comparison of training overhead

The random workload was used to comparatively evaluate the overhead of training time spent in the total execution time. The total execution time is compared after normalization based on the SJF, which has the shortest execution time. Assuming that all applications' resource utilization histories exist, the total execution time including the training time for the first job placement is compared. In the cases of DqnGPU, DeepRM Max, and DeepRM Mean method, the training time is included because job placement is performed based on the results obtained through the reinforcement learning.

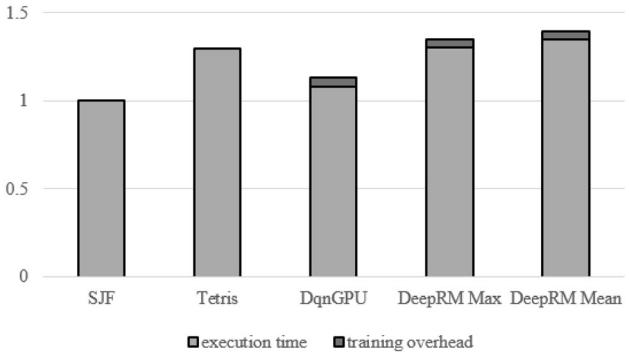


Fig. 9 Comparison of training overheads

Figure 9 shows the training overhead results of the respective placement methods. The normalized values of the execution time for Tetris, the proposed method of this study, DeepRM Max, and DeepRM Mean is approximately 1.294, 1.081, 1.303, and 1.346, respectively. The proposed methods' training time is approximately 0.054 and the execution time minus the training time is 1.027. When the result of the execution time is compared with that of SJF, the difference in performance is small. This implies that except for the first training time of job set, there is almost difference of performance in the case of placing a next set of jobs thereafter. This means that the performance of the next job set can be improved because the accuracy as well the resource utilization will increase when the training is continually performed online while performing the jobs. The training time of DeepRM Max is 0.048, and when only the execution time is compared between Tetris (1.294) and DeepRM Max (1.254), the latter is superior. Therefore, the job placement method that applies the reinforcement learning excluding the first training overhead is significant.

4.2.4 SJF vs DQNGPU about cost

Depending on the workload or executed application, the performance may be better when it is placed and executed alone. However, the earlier experiments confirm that co-placement is superior for every type of workload in the resource utilization aspect. This implies that there is an advantage with respect to the operation of clusters. In order to explain the advantage of co-placements of resources in the user aspect, the cost was compared between the SJF and DQNPGPU based on the GPU instance provided by a cloud company. Because the cost of the GPU-sharing instance does not exist, the instance costs were compared based on similar GPU cards currently provided. Table 4 shows the GPU instance costs provided by Amazon EC2. G3 instance and G4 instance provide NVIDIA Tesla M60 and NVIDIA

Table 4 Amazon EC2's GPU instance costs

Name	GPU	vCPU	Mem.(GB)	GPU Mem. (GB)	GPU cores	Cost ($)
g3.xlarge	1	4	30.5	8	2048	0.75
g4dn. xlarge	1	4	16	16	2560	0.528
g4dn. 4xlarge	1	16	64	16	2560	1.204

T4 GPU cards, respectively. In the aspect of Cost 1: Regarding the GPU memory resource, suppose g4dn.4xlarge is the same as two units of g3.xlarge; then, $0.602 can be saved when g4dn.4xlarge is shared. Furthermore, in the aspect of Cost 2: Regarding vCPU and memory resources, supposing g4dn.4xlarge is same as four units of g4dn.xlarge 4; then, up to $0.301 in cost can be saved when g4dn.4xlarge is shared.

The prices above were compared based on the cost of executing the random workload. Figure 10 compares the costs of SJF and DQNGPU. In the case of Cost 1, the cost was reduced by about 29.86% when the co-placement instance was used as compared with using a single placement instance. In the case of Cost 2, the cost was reduced by about 49.98% when a resource was shared as compared with using a single instance.

5 Related works

This section describes the resource management method based on resource sharing in the GPU cluster environment. Furthermore, studies on various resource scheduling methods using machine-learning are introduced, and their results are compared with this study in Table 5.

5.1 GPU sharing

As GPUs have become increasingly widely used, cloud providers are providing various kinds of GPU resources. The conventional service method, which allocates just one user to a node, can lead to resource waste of GPU node and have a relatively high cost. Therefore, studies have been conducted on methods of sharing and managing GPU resources, with the goal of effectively using the resources in the cloud and cluster environment.

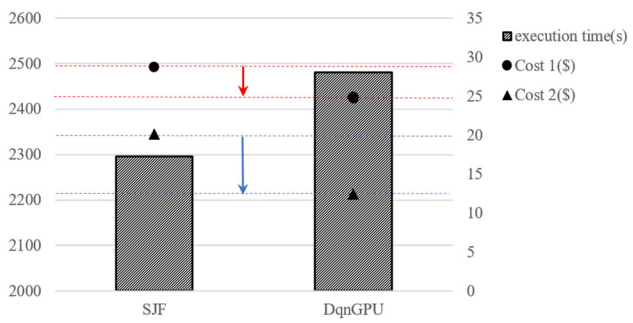


Fig. 10 Cost comparison: SJF vs DQNGPU

Hong [16] proposed a system for fair sharing by calculating the weight according to the GPU utilization intensity of HPC application programs in a cloud environment. When the GPU kernel execution request is short and repetitive, an overhead of memory mapping I/O, etc. occurs. This issue is solved by managing the queue without lock by using the shared memory region between virtual machines (VMs). However, the fairness management of the resource use between VMs cannot be properly performed when the number and length of the kernels are vary, as for HPC applications. Diab [35] proposed a system that facilitated the simultaneous execution of jobs by sharing GPU resources among different users in a cluster environment. Using an openstack-based VM, each resource's information and executed application's information were collected, and based on these, an API was seized, and two GPU kernels could be executed. Execution time and GPU memory information was collected for the information of the application, and in the experiment, the scheduling was performed for only the machine learning applications which have compute-intensive characteristics. However, for the case of HPC applications, as they can have I/O intensive characteristics, additional information of shared resources and monitored resources is need.

5.2 System management using ML

Machine learning studies have been actively performed for system operations in diverse areas such as purpose classification of load balancing, traffic routing, and patch files [36]. Various past and current data such as environment, application information, and environmental change according to executed application can be received as inputs, and based on these, the results such as interference between applications, job placement, and resource management can be obtained through learning. Hence, interest is increasing in the application of machine learning to cluster operation [37], and studies have been carried out to improve job and resource management [38–42].

Rossi [38] proposed a resource management technique capable of auto-scaling based on the reinforcement learning among the machine learning methods. Optimal horizontal scaling and vertical scaling of the next job container are decided by using the dynamic-Q in the reinforcement

Table 5 ML-based scheduling features comparison

Features	[39]	[40]	[42]	[41]	DqnGPU
ML method	DQN	DRL	CF	RF, LR	DQN
Target resource	CPU	GPU	GPU	GPU	GPU
Target apps.	Virtual job	ML	ML	ML	ML & HPC
Execution/non-execution of multiple apps.	x	o (multi-apps)	o(3 apps)	o (2 apps)	o (multi-apps)
Consideration/non-consideration of interference-related detailed	x	x	x	o	o

learning. The learning was performed based on the CPU occupancy history of applications with the goal of strategically minimizing the runtime cost. Mao [39] proposed a resource management technology based on the Deep Q-Networks (DQN) reinforcement learning. The CPU and the memory were divided into certain slot units according to the execution time, and they were used as input values to perform the learning. Based on the learning, the resource provision for the next job are generated as a reward value, and this is reflected in the environment. However, because the above studies conducted the learning based on the CPU resource history, additional resources (GPU memory, PCIe, etc.) should be considered when executed on the GPU resource.

Bao [40] proposed a job placement framework using deep reinforcement learning (DRL) in the GPU cluster environment. The learning model learns a total of three inputs: user, CPU, and GPU resources. Based on this, jobs that will be simultaneously executed on the server are placed. The goal of the reward is to minimize the average job completion time. The study of [40] states that jobs are placed with a low level of interference when different types of machine learning applications exist. However, the CPU and GPU histories of applications are not adequate enough to reduce the effects of interference, and more inputs are needed for other resources, on which interference may occur. Ukidave [41] proposed an interference-aware scheduler for applications based on a machine learning model through the history values of applications executed on the GPU. The lengths of kernels were changed to investigate the histories of interference with each other, but it was proven that they were not actually applicable to the applications. Accordingly, the histories, in which interference may occur between applications, were defined, and learned through Random Forest (RF) and Linear Regression (LR), which are regression models. However, because the histories defined in the paper were learned by focusing on the length of the kernel, they were suitable only for the machine learning applications having a certain kernel execution pattern. The study of [42] proposed an interference-aware scheduler for co-execution of applications on the GPU-based cluster and cloud server. Only simple histories are profiled online, and the prediction is performed based on the empty Single Value Decomposition's (SVD's) table values using collaborative filtering (CF). However, in the case of collaborative filtering, the result can converge to an incorrect prediction value. Furthermore, this method is appropriate in the case of an application having a certain characteristic but it may not be appropriate for an application for which prediction is impossible.

6 Conclusion

This study proposed a reinforcement learning DQN-based job placement method that reflects the job history of applications in the GPU cluster environment. Reinforcement learning performs the learning through the DNN which reflects the job history, including changing resource utilization, and obtains the multiple job placement action values. Based on the proposed data placement method, co-placement and resource sharing were facilitated targeting the applications with various characteristics. The workloads classified by the characteristics were generated and the comparison target methods, i.e., a single placement method, multiple placement methods, and the reinforcement learning-applied multiple placement method were compared. The experimental results confirmed that the proposed method constituted a significant improvement, as it achieved low performance degradation and better resource utilization. Furthermore, the training overhead that might occur was analyzed and the results proved that its impact on the total performance was small. In addition, the benefits of GPU sharing placements were explained by comparing the GPU resource cost between a single placement and the sharing placement. In a future study, a cluster resource environment will be added, and the resource environment of executed applications will be changed in order to consider the changed job history in the reinforcement learning. Furthermore, the method will be expanded to include offline job placement for the first

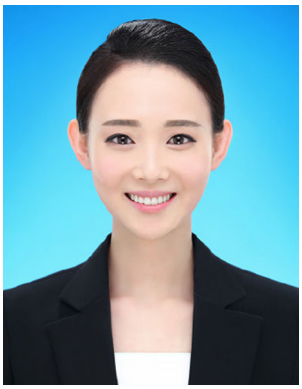
placement in order to reduce the training overhead when first executed.

Acknowledgements The authors would like to thank all students who contributed to this study. We are grateful to Qichen Chen, Sejin Kim, who assisted with evaluation.This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (Nos. NRF-2015M3C4A7065646,2017R1A2B4005681).

References

1. Amazon ec2. https://aws.amazon.com/ec2/
2. Nimbix. https://www.nimbix.net/cloud-computing-nvidia/
3. Microsoft azure. https://docs.microsoft.com/en-au/azure/virtual-machines/windows/sizes-gpu
4. Alibaba. https://www.alibabacloud.com/ko/product/gpu
5. Kubernetes. https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/
6. Mesos. http://mesos.apache.org/documentation/latest/gpu-support/
7. Liu, M., Li, T., Jia, N., Currid, A., Troy, V.: Understanding the virtualization" tax" of scale-out pass-through gpus in gaas clouds: An empirical study. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pp. 259–270. IEEE (2015)
8. Tang, X., Wang, H., Ma, X., El-Sayed, N., Zhai, J., Chen, W., Aboulnaga, A.: Spread-n-share: improving application performance and cluster throughput with resource-aware job placement. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 12. ACM (2019)
9. Duato, J., Pena, A.J., Silla, F., Mayo, R., Quintana-Ortí, E.S.: rCUDA: Reducing the number of gpu-based accelerators in high performance clusters. In: 2010 International Conference on High Performance Computing & Simulation, pp. 224–231. IEEE (2010)
10. Ilager, S., Wankar, R., Kune, R., Buyya, R.: Gpu paas computation model in aneka cloud computing environments. Smart Data: State-of-the-Art Perspectives in Computing and Applications, p. 19 (2019)
11. Toosi, A.N., Sinnott, R.O., Buyya, R.: Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using aneka. Future Gener. Comput. Syst. **79**, 765–775 (2018)
12. Mps. https://docs.nvidia.com/deploy/mps/index.html
13. Chang, C.-C., Yang, S.-R., Yeh, E.-H., Lin, P., Jeng, J.-Y.: A kubernetes-based monitoring platform for dynamic cloud resource provisioning. In: GLOBECOM 2017-2017 IEEE Global Communications Conference, pp. 1–6. IEEE (2017)
14. Gu, J., Song, S., Li, Y., Luo, H., Gaiagpu: Sharing gpus in container clouds. In: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), pp. 469–476. IEEE (2018)
15. Song, S., Deng, L., Gong, J., Luo, H.: Gaia scheduler: A kubernetes-based scheduler framework. In: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), pp. 252–259. IEEE (2018)
16. Hong, C.-H., Spence, I., Nikolopoulos, D.S.: Fairgv: fair and fast gpu virtualization. IEEE Trans. Parallel Distrib. Syst. **28**(12), 3472–3485 (2017)
17. Tanasic, I., Gelado, I., Cabezas, J., Ramirez, A., Navarro, N., Valero, M.: Enabling preemptive multiprogramming on gpus. In 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), pp. 193–204. IEEE (2014)
18. Ukidave, Y., Kalra, C., Kaeli, D., Mistry, P., Schaa, D.: Runtime support for adaptive spatial partitioning and inter-kernel communication on gpus. In: 2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing, pp. 168–175. IEEE (2014)
19. Li, X., Zhang, G., Howie Huang, H., Wang, Z., Zheng, W.: Performance analysis of gpu-based convolutional neural networks. In: 2016 45th International Conference on Parallel Processing (ICPP), pp. 67–76. IEEE (2016)
20. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283 (2016)
21. Lammps. https://lammps.sandia.gov/
22. Qmcpack. https://qmcpack.org/
23. Phull, R., Li, C.-H., Rao, K., Cadambi, H., Chakradhar, S.: Interference-driven resource management for gpu-based heterogeneous clusters. In: Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, pp. 109–120. ACM (2012)
24. Nvidia gpu cloud. https://ngc.nvidia.com/
25. Dutreilh, X., Kirgizov, S., Melekhova, O., Malenfant, J., Rivierre, N., Truck, I.: Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In: ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems, pp. 67–74 (2011)
26. Barrett, E., Howley, E., Duggan, J.: Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. Concurr. Comput. Pract. Exp. **25**(12), 1656–1674 (2013)
27. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)
28. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, (2013)
29. deeprm. https://github.com/hongzimao/deeprm
30. Gromacs. http://www.gromacs.org/
31. Hoomd. http://glotzerlab.engin.umich.edu/hoomd-blue/
32. Alibaba fake gpu. https://github.com/AliyunContainerService/gpushare-scheduler-extender
33. Nvidia docker container. https://github.com/NVIDIA/nvidia-docker
34. Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S., Akella, A.: Multi-resource packing for cluster schedulers. In: ACM SIGCOMM Computer Communication Review, vol. 44, pp. 455–466. ACM (2014)
35. Diab, K.M., Mustafa Rafique, M., Hefeeda, M.: Dynamic sharing of gpus in cloud systems. In: 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, pp. 947–954. IEEE (2013)
36. Lawall, Levin S.: J. Building stable kernel trees with machine learning
37. 2019 usenix: Conference on operational machine learning. https://www.usenix.org/conference/opml19
38. Rossi, F., Nardelli, M., Cardellini, V.: Horizontal and vertical scaling of container-based applications using reinforcement

learning. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pp. 329–338. IEEE (2019)
39. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, pp. 50–56. ACM (2016)
40. Bao, Y., Peng, Y., Wu, C.: Deep learning-based job placement in distributed machine learning clusters. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 505–513. IEEE (2019)
41. Xu, X., Zhang, N., Cui, M., He, M., Surana, R.: Characterization and prediction of performance interference on mediated pass-through gpus for interference-aware scheduler. In: 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19), (2019)
42. Ukidave, Y., Li, X., Kaeli, D.: Mystic: Predictive scheduling for gpu based cloud servers using machine learning. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 353–362. IEEE (2016)



Jisun Oh She is currently a Master's student in the Department of Computer Science, Sookmyung Women's University. She received her B.S. degree from Sookmyung Women's University in 2017. She is also researcher at the Distributed & Cloud Computing Lab of Sookmyung Women's University. Her research interests include cloud computing and management in distributed computing systems.

Yoonhee Kim she is the professor of Computer Science Department at Sookmyung Women's University. She received her Bachelors degree from Sookmyung Women's University in 1991, her Master degree and Ph.D. from Syracuse University in 1996 and 2001, respectively. She was a Research Staff Member at the Electronics and Telecommunication Research Institute during 1991 and 1994. Before joining the faculty of Sookmyung Women's University in 2001, she was the faculty of Computer Engineering dept. at Rochester Institute of Technology in NY, USA. Her research interests span many aspects of runtime support and management in distributed computing systems. She is a member of IEEE and OGF, and she has served on variety of program committees, advisory boards, and editorial boards.